

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

January 25, 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`).

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

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

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

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

## 1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

We load some packages.

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

---

\*This document corresponds to the version 6.26d of `nicematrix`, at the date of 2024/01/25.

```

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

```

With Overleaf, by default, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key H in order to have more information. That's why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key `messages-for-Overleaf` is used (at load-time).

```

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

```

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

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \bool_if:NTF \g_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }

```

We try to detect whether the compilation is done on Overleaf. We use `\c_sys_jobname_str` because, with Overleaf, the value of `\c_sys_jobname_str` is always “output”.

```

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

```

```

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

```

## 2 Security test

Within the package `nicematrix`, we will have to test whether a cell of a `{NiceTabular}` is empty. For the cells of the columns of type p, b, m, X and V, we will test whether the cell is syntactically empty

(that is to say that there is only spaces between the ampersands &). That test will be done with the command `\@@_test_if_empty:` by testing if the two first tokens in the cells are (during the TeX process) are `\ignorespaces` and `\unskip`.

However, if, one day, there is a changement in the implementation of `array`, maybe that this test will be broken (and `nicematrix` also).

That's why, by security, we will take a test in a small `{tabular}` composed in the box `\l_tmpa_box` used as sandbox.

```

52 \@@_msg_new:nn { Internal~error }
53 {
54   Potential~problem~when~using~nicematrix.\\
55   The~package~nicematrix~have~detected~a~modification~of~the~
56   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
57   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
58   this~message~again,~load~nicematrix~with:~\token_to_str:N
59   \usepackage[no-test-for-array]{nicematrix}.
60 }

61 \@@_msg_new:nn { mdwtab~loaded }
62 {
63   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
64   This~error~is~fatal.
65 }

66 \cs_new_protected:Npn \@@_security_test:n #1
67 {
68   \peek_meaning:NTF \ignorespaces
69   { \@@_security_test_i:w }
70   { \@@_error:n { Internal~error } }
71   #1
72 }

73 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
74 {
75   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
76   #1
77 }

```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test. This code has been modified in version 6.18 (see question 682891 on TeX StackExchange).

```

78 \hook_gput_code:nnn { begindocument / after } { . }
79 {
80   \IfPackageLoadedTF { mdwtab }
81   { \@@_fatal:n { mdwtab~loaded } }
82   {
83     \bool_if:NF \g_@@_no_test_for_array_bool
84     {
85       \group_begin:
86       \hbox_set:Nn \l_tmpa_box
87       {
88         \begin { tabular } { c > { \@@_security_test:n } c c }
89         text & & text
90         \end { tabular }
91       }
92       \group_end:
93     }
94   }
95 }

```

### 3 Collecting options

The following technic allows to create user commands with the ability to put an arbitrary number of `[list of (key=val)]` after the name of the command.

*Exemple :*

`\@@_collect_options:n { \F } [x=a,y=b] [z=c,t=d] { arg }`

will be transformed in : `\F{x=a,y=b,z=c,t=d}{arg}`

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,

the command `\G` takes in an arbitrary number of optional arguments between square brackets.

Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

96 \cs_new_protected:Npn \@@_collect_options:n #1
97 {
98   \peek_meaning:NTF [
99     { \@@_collect_options:nw { #1 } }
100     { #1 { } }
101   }

```

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

```

102 \NewDocumentCommand \@@_collect_options:nw { m r[] }
103 { \@@_collect_options:nn { #1 } { #2 } }
104
105 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
106 {
107   \peek_meaning:NTF [
108     { \@@_collect_options:nnw { #1 } { #2 } }
109     { #1 { #2 } }
110   }
111
112 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
113 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

### 4 Technical definitions

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

```

114 \tl_const:Nn \c_@@_b_tl { b }
115 \tl_const:Nn \c_@@_c_tl { c }
116 \tl_const:Nn \c_@@_l_tl { l }
117 \tl_const:Nn \c_@@_r_tl { r }
118 \tl_const:Nn \c_@@_all_tl { all }
119 \tl_const:Nn \c_@@_dot_tl { . }
120 \tl_const:Nn \c_@@_default_tl { default }
121 \tl_const:Nn \c_@@_star_tl { * }
122 \str_const:Nn \c_@@_r_str { r }
123 \str_const:Nn \c_@@_c_str { c }
124 \str_const:Nn \c_@@_l_str { l }
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_@@_j_str { j }
129 \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).

```

130 \tl_new:N \l_@@_argspec_tl
131 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
132 \cs_generate_variant:Nn \str_lowercase:n { V }

133 \hook_gput_code:nnn { begindocument } { . }
134 {
135   \IfPackageLoadedTF { tikz }
136   {

```

In some constructions, we will have to use a `{pgfpicture}` which *must* be replaced by a `{tikzpicture}` if Tikz is loaded. However, this switch between `{pgfpicture}` and `{tikzpicture}` can't be done dynamically with a conditional because, when the Tikz library `external` is loaded by the user, the pair `\tikzpicture-\endtikzpicture` (or `\begin{tikzpicture}-\end{tikzpicture}`) must be statically “visible” (even when externalization is not activated).

That's why we create `\c_@@_pgfortikzpicture_tl` and `\c_@@_endpgfortikzpicture_tl` which will be used to construct in a `\AtBeginDocument` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

137   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
138   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
139 }
140 {
141   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
142   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
143 }
144 }

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date May 2023, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

145 \IfClassLoadedTF { revtex4-1 }
146 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
147 {
148   \IfClassLoadedTF { revtex4-2 }
149   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
150   {

```

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

```

151   \cs_if_exist:NT \rvtx@ifformat@geq
152   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
153   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
154 }
155 }

```

```

156 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

If the final user uses `nicematrix`, PGF/Tikz will write instruction `\pgfsyspdfmark` in the `aux` file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the `aux` file. With the following code, we try to avoid that situation.

```

157 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
158 {
159   \iow_now:Nn \@mainaux
160   {
161     \ExplSyntaxOn
162     \cs_if_free:NT \pgfsyspdfmark
163     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
164     \ExplSyntaxOff

```

```

165     }
166     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
167 }

```

We define a command `\iddots` similar to `\ddots` (`\ddots`) but with dots going forward (`\iddots`). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package `mathdots`), we don't define it again.

```

168 \ProvideDocumentCommand \iddots { }
169 {
170   \mathinner
171   {
172     \tex_mkern:D 1 mu
173     \box_move_up:nn { 1 pt } { \hbox { . } }
174     \tex_mkern:D 2 mu
175     \box_move_up:nn { 4 pt } { \hbox { . } }
176     \tex_mkern:D 2 mu
177     \box_move_up:nn { 7 pt }
178     { \vbox:n { \kern 7 pt \hbox { . } } }
179     \tex_mkern:D 1 mu
180   }
181 }

```

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.

```

182 \hook_gput_code:nnn { begindocument } { . }
183 {
184   \IfPackageLoadedTF { booktabs }
185   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
186   { }
187 }
188 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
189 {
190   \cs_set_eq:NN \@@_old_pgfutil@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`).

```

191   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
192   {
193     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
194     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
195   }
196 }

```

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

```

197 \hook_gput_code:nnn { begindocument } { . }
198 {
199   \IfPackageLoadedTF { colortbl }
200   { }
201   {

```

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.

```

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

```

Idem for \CT@drs@.

```

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

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.

```

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

```

The following \skip\_horizontal:N \c\_zero\_dim is to prevent a potential \unskip to delete the \leaders<sup>1</sup>

```

237     \skip_horizontal:N \c_zero_dim
238 }
```

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.

```

239     \everycr { }
240     \cr
241     \noalign { \skip_vertical:N -\arrayrulewidth }
242 }
```

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.

```

243 \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.

```

244 { \@@_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*.

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
247 {
```

---

<sup>1</sup>See question 99041 on TeX StackExchange.

```

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

```

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).

```

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

```

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

```

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

```

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

```

269 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

270 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
271 {
272 \tl_if_blank:nF { #1 }
273 {
274 \tl_if_head_eq_meaning:nNTF { #1 } [
275 { \cs_set:Npn \CT@arc@ { \color #1 } }
276 { \cs_set:Npn \CT@arc@ { \color { #1 } } }
277 ]
278 }
279 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }

```

```

280 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
281 {
282 \tl_if_head_eq_meaning:nNTF { #1 } [
283 { \cs_set:Npn \CT@drsc@ { \color #1 } }
284 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
285 ]
286 \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.

```

287 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
288 {
289 \tl_if_head_eq_meaning:nNTF { #2 } [
290 { #1 #2 }
291 { #1 { #2 } }
292 ]
293 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

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

297 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

298 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
299   {
300     \tl_set_rescan:Nno
301       #1
302       {
303         \char_set_catcode_other:N >
304         \char_set_catcode_other:N <
305       }
306     #1
307   }

```

## 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.

```

308 \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).

```

309 \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.

```

310 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
311   { \int_use:N \g_@@_env_int }

```

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

```

312 \cs_new_protected:Npn \@@_qpoint:n #1
313   { \pgfpaintanchor { \@@_env: - #1 } { center } }

```

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

```

314 \bool_new:N \l_@@_tabular_bool

```

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

```

315 \bool_new:N \g_@@_delims_bool
316 \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.

```
317 \bool_new:N \l_@@_preamble_bool
318 \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.

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

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

```
320 \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.

```
321 \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).

```
322 \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.).

```
323 \dim_new:N \l_@@_col_width_dim
324 \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.

```
325 \int_new:N \g_@@_row_total_int
326 \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).

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

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

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

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

```
329 \tl_new:N \l_@@_hpos_cell_tl
330 \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`.

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

Idem for the mono-row blocks.

```
332 \dim_new:N \g_@@_blocks_ht_dim
333 \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}`).

```
334 \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`.

```
335 \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.

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

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

```
337 \bool_new:N \l_@@_notes_detect_duplicates_bool
338 \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.

```
339 \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).

```
340 \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).

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

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

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

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

```
343 \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.

```
344 \bool_new:N \l_@@_X_bool
```

```
345 \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 }`).

```
346 \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.

```
347 \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.

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

349 \tl_new:N \g_@@_left_delim_tl
350 \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}`).

```
351 \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`).

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

For `\multicolumn`.

```
353 \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`.

```
354 \tl_new:N \l_@@_columns_type_tl
355 \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 `:`.

```
356 \tl_new:N \l_@@_xdots_down_tl
357 \tl_new:N \l_@@_xdots_up_tl
358 \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`).

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

360 \cs_new_protected:Npn \@@_test_if_math_mode:
361 {
362   \if_mode_math: \else:
363     \@@_fatal:n { Outside-math-mode }
364   \fi:
365 }
```

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.

```
366 \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”.

```
367 \colorlet { nicematrix-last-col } { . }
368 \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`).

```
369 \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*.

```
370 \tl_new:N \g_@@_com_or_env_str
371 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
372 \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).

```
373 \cs_new:Npn \g_@@_full_name_env:
374 {
375   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
376   { command \space \c_backslash_str \g_@@_name_env_str }
377   { environment \space \{ \g_@@_name_env_str \} }
378 }
```

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

```
379 \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.

```
380 \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.

```
381 \tl_new:N \g_@@_pre_code_before_tl
382 \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.

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

```
385 \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.

```
386 \int_new:N \l_@@_old_iRow_int
387 \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).

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

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

```
389 \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.

```
390 \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.

```
391 \bool_new:N \l_@@_X_columns_aux_bool
392 \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.

```
393 \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).

```
394 \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).

```
395 \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`).

```
396 \tl_new:N \l_@@_code_before_tl
397 \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).

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

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

```
399 \dim_new:N \l_@@_x_initial_dim
400 \dim_new:N \l_@@_y_initial_dim
401 \dim_new:N \l_@@_x_final_dim
402 \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.

```
403 \dim_zero_new:N \l_@@_tmpc_dim
404 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
405 \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”.

```
406 \dim_new:N \g_@@_width_last_col_dim
407 \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.

```
408 \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!).

```
409 \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}`.

```
410 \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.

```
411 \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.

```
412 \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`).

```
413 \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.

```
414 \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).

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

```
416 \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`).

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

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

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

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

The following counters will be used when drawing the rules.

```

421 \int_new:N \l_@@_start_int
422 \int_set_eq:NN \l_@@_start_int \c_one_int
423 \int_new:N \l_@@_end_int
424 \int_new:N \l_@@_local_start_int
425 \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.

```

426 \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).

```

427 \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`.

```

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

```

434 \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`.

```

435 \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.

```

436 \dim_new:N \l_@@_offset_dim

```

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

```

437 \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`).

```

438 \str_new:N \l_@@_hpos_block_str
439 \str_set:Nn \l_@@_hpos_block_str { c }
440 \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.

```

441 \bool_new:N \@@_nocolor_used_bool

```

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

```

442 \str_new:N \l_@@_vpos_block_str
443 \str_set:Nn \l_@@_vpos_block_str { c }

```

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

```
444 \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).

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

```
446 \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.

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

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

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

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

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

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

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

```
454 \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}`).

```
455 \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.

```
456 \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`).

```
457 \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.

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

```
459 \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.

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

```
461 \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).

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

```
463 \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”.<sup>2</sup>

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

Idem for `\l_@@_last_col_without_value_bool`

```
465 \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`.

```
466 \int_new:N \l_@@_last_col_int
467 \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.

```
468 \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`).

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

### Some utilities

```
470 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
471 {
472   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
473   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
474 }
```

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.

```
475 \cs_new_protected:Npn \@@_expand_clist:N #1
476 {
477   \clist_if_in:NVF #1 \c_@@_all_tl
478   {
479     \clist_clear:N \l_tmpa_clist
480     \clist_map_inline:Nn #1
```

---

<sup>2</sup>We 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.

```

481     {
482       \tl_if_in:nnTF { ##1 } { - }
483       { \@@_cut_on_hyphen:w ##1 \q_stop }
484       {
485         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
486         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
487       }
488       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
489       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
490     }
491     \tl_set_eq:NN #1 \l_tmpa_clist
492   }
493 }

```

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.

```

494 \hook_gput_code:nnn { begindocument } { . }
495 {
496   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
497   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
498   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
499 }

```

## 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`.<sup>3</sup>
  - 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`).

---

<sup>3</sup>More 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.

```

500 \newcounter { tabularnote }
501 \seq_new:N \g_@@_notes_seq
502 \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.

```

503 \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.

```

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

```

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

```

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

```

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

```

512 \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`.

```

513 \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`.

```

514 \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`).

```

515 \hook_gput_code:nnn { begindocument } { . }
516 {
517   \IfPackageLoadedTF { enumitem }
518   {

```

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.

```

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

```

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`.

```

538     \NewDocumentCommand \tabularnote { o m }
539     {
540         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
541         {
542             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
543             { \@@_error:n { tabularnote~forbidden } }
544             {
545                 \bool_if:NTF \l_@@_in_caption_bool
546                 \@@_tabularnote_caption:nn
547                 \@@_tabularnote:nn
548                 { #1 } { #2 }
549             }
550         }
551     }
552 }
553 {
554     \NewDocumentCommand \tabularnote { o m }
555     {
556         \@@_error_or_warning:n { enumitem~not~loaded }
557         \@@_gredirect_none:n { enumitem~not~loaded }
558     }
559 }
560 }

561 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
562 { \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`.

```

563 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
564 {

```

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).

```

565     \int_zero:N \l_tmpa_int
566     \bool_if:NT \l_@@_notes_detect_duplicates_bool
567     {

```

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`.

```

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

```

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`.

```

606     \hbox_set:Nn \l_tmpa_box
607     {

```

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

```

608         \@@_notes_label_in_tabular:n
609         {

```

```

610         \seq_use:Nnnn
611         \l_@@_notes_labels_seq { , } { , } { , }
612     }
613 }

```

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

```

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

```

625         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
626     }
627     { \box_use:N \l_tmpa_box }
628 }
629 }

```

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`.

```

630 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
631 {
632     \bool_if:NTF \g_@@_caption_finished_bool
633     {
634         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
635         { \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`!

```

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

```

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

```

640     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
641     {

```

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.

```

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

```

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!

```

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

662 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
663 { \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.

```

664 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
665 {
666   \begin { pgfscope }
667   \pgfset
668   {
669     inner~sep = \c_zero_dim ,
670     minimum~size = \c_zero_dim
671   }
672   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
673   \pgfnode
674   { rectangle }
675   { center }
676   {
677     \vbox_to_ht:nn
678     { \dim_abs:n { #5 - #3 } }
679     {
680       \vfill
681       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
682     }
683   }
684   { #1 }
685   { }
686   \end { pgfscope }
687 }

```

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.

```

688 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
689 {
690   \begin { pgfscope }
691   \pgfset
692   {
693     inner~sep = \c_zero_dim ,
694     minimum~size = \c_zero_dim

```

```

695     }
696     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
697     \pgfpointdiff { #3 } { #2 }
698     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
699     \pgfnode
700     { rectangle }
701     { center }
702     {
703         \vbox_to_ht:nn
704         { \dim_abs:n \l_tmpb_dim }
705         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
706     }
707     { #1 }
708     { }
709     \end { pgfscope }
710 }

```

## 8 The options

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

```

711 \tl_new:N \l_@@_caption_tl
712 \tl_new:N \l_@@_short_caption_tl
713 \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).

```

714 \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.

```

715 \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`.

```

716 \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`).

```

717 \dim_new:N \l_@@_cell_space_top_limit_dim
718 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

719 \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.

```

720 \dim_new:N \l_@@_xdots_inter_dim
721 \hook_gput_code:nnn { begindocument } { . }
722 { \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).

```

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

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.

```

730 \dim_new:N \l_@@_xdots_radius_dim
731 \hook_gput_code:nnn { begindocument } { . }
732 { \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.

```

733 \tl_new:N \l_@@_xdots_line_style_tl
734 \tl_const:Nn \c_@@_standard_tl { standard }
735 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```

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

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

```

737 \tl_new:N \l_@@_baseline_tl
738 \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`).

```

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

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

```

740 \bool_new:N \l_@@_parallelize_diags_bool
741 \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.

```

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

```

743 \dim_new:N \l_@@_notes_above_space_dim
744 \hook_gput_code:nnn { begindocument } { . }
745 { \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.).

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

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

```
747 \cs_new_protected:Npn \@@_reset_arraystretch:
748 { \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).

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

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

```
750 \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.

```
751 \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”.

```
752 \bool_new:N \l_@@_medium_nodes_bool
753 \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).

```
754 \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`).

```
755 \dim_new:N \l_@@_left_margin_dim
756 \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`.

```
757 \dim_new:N \l_@@_extra_left_margin_dim
758 \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.

```
759 \tl_new:N \l_@@_end_of_row_tl
760 \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 “:”.

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

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

```
762 \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.

```

763 \bool_new:N \l_@@_delimiters_max_width_bool

764 \keys_define:nn { NiceMatrix / xdots }
765 {
766   shorten-start .code:n =
767     \hook_gput_code:nnn { begindocument } { . }
768     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
769   shorten-end .code:n =
770     \hook_gput_code:nnn { begindocument } { . }
771     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
772   shorten-start .value_required:n = true ,
773   shorten-end .value_required:n = true ,
774   shorten .code:n =
775     \hook_gput_code:nnn { begindocument } { . }
776     {
777       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
778       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
779     } ,
780   shorten .value_required:n = true ,
781   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
782   horizontal-labels .default:n = true ,
783   line-style .code:n =
784     {
785       \bool_lazy_or:nnTF
786         { \cs_if_exist_p:N \tikzpicture }
787         { \str_if_eq_p:nn { #1 } { standard } }
788         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
789         { @@_error:n { bad-option-for~line-style } }
790     } ,
791   line-style .value_required:n = true ,
792   color .tl_set:N = \l_@@_xdots_color_tl ,
793   color .value_required:n = true ,
794   radius .code:n =
795     \hook_gput_code:nnn { begindocument } { . }
796     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
797   radius .value_required:n = true ,
798   inter .code:n =
799     \hook_gput_code:nnn { begindocument } { . }
800     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
801   radius .value_required:n = true ,

```

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

```

802   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
803   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
804   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).

```

805   draw-first .code:n = \prg_do_nothing: ,
806   unknown .code:n = @@_error:n { Unknown~key~for~xdots }
807 }

```

```

808 \keys_define:nn { NiceMatrix / rules }
809 {
810   color .tl_set:N = \l_@@_rules_color_tl ,
811   color .value_required:n = true ,
812   width .dim_set:N = \arrayrulewidth ,
813   width .value_required:n = true ,
814   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
815 }

```

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

```

816 \keys_define:nn { NiceMatrix / Global }
817 {
818   no-cell-nodes .code:n =
819     \cs_set_protected:Npn \@@_node_for_cell:
820     { \box_use_drop:N \l_@@_cell_box } ,
821   no-cell-nodes .value_forbidden:n = true ,
822   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
823   rounded-corners .default:n = 4 pt ,
824   custom-line .code:n = \@@_custom_line:n { #1 } ,
825   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
826   rules .value_required:n = true ,
827   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
828   standard-cline .default:n = true ,
829   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
830   cell-space-top-limit .value_required:n = true ,
831   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
832   cell-space-bottom-limit .value_required:n = true ,
833   cell-space-limits .meta:n =
834   {
835     cell-space-top-limit = #1 ,
836     cell-space-bottom-limit = #1 ,
837   } ,
838   cell-space-limits .value_required:n = true ,
839   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
840   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
841   light-syntax .default:n = true ,
842   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
843   end-of-row .value_required:n = true ,
844   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
845   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
846   last-row .int_set:N = \l_@@_last_row_int ,
847   last-row .default:n = -1 ,
848   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
849   code-for-first-col .value_required:n = true ,
850   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
851   code-for-last-col .value_required:n = true ,
852   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
853   code-for-first-row .value_required:n = true ,
854   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
855   code-for-last-row .value_required:n = true ,
856   hlines .clist_set:N = \l_@@_hlines_clist ,
857   vlines .clist_set:N = \l_@@_vlines_clist ,
858   hlines .default:n = all ,
859   vlines .default:n = all ,
860   vlines-in-sub-matrix .code:n =
861   {
862     \tl_if_single_token:nTF { #1 }
863     {
864       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
865       { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

866         { \cs_set_eq:cn { @@ _ #1 } \@@_make_preamble_vlism:n }
867     }
868     { \@@_error:n { One~letter~allowed } }
869 },
870 vlines-in-sub-matrix .value_required:n = true ,
871 hvlines .code:n =
872 {
873     \bool_set_true:N \l_@@_hvlines_bool
874     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
875     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
876 },
877 hvlines-except-borders .code:n =
878 {
879     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
880     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
881     \bool_set_true:N \l_@@_hvlines_bool
882     \bool_set_true:N \l_@@_except_borders_bool
883 },
884 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.

```

885 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
886 renew-dots .value_forbidden:n = true ,
887 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
888 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
889 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
890 create-extra-nodes .meta:n =
891 { create-medium-nodes , create-large-nodes } ,
892 left-margin .dim_set:N = \l_@@_left_margin_dim ,
893 left-margin .default:n = \arraycolsep ,
894 right-margin .dim_set:N = \l_@@_right_margin_dim ,
895 right-margin .default:n = \arraycolsep ,
896 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
897 margin .default:n = \arraycolsep ,
898 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
899 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
900 extra-margin .meta:n =
901 { extra-left-margin = #1 , extra-right-margin = #1 } ,
902 extra-margin .value_required:n = true ,
903 respect-arraystretch .code:n =
904     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
905 respect-arraystretch .value_forbidden:n = true ,
906 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
907 pgf-node-code .value_required:n = true
908 }

```

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

```

909 \keys_define:nn { NiceMatrix / Env }
910 {
911     corners .clist_set:N = \l_@@_corners_clist ,
912     corners .default:n = { NW , SW , NE , SE } ,
913     code-before .code:n =
914     {
915         \tl_if_empty:nF { #1 }
916         {
917             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
918             \bool_set_true:N \l_@@_code_before_bool
919         }
920     } ,
921     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}`.

```

922 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
923 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
924 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
925 baseline .tl_set:N = \l_@@_baseline_tl ,
926 baseline .value_required:n = true ,
927 columns-width .code:n =
928   \tl_if_eq:nnTF { #1 } { auto }
929   { \bool_set_true:N \l_@@_auto_columns_width_bool }
930   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
931 columns-width .value_required:n = true ,
932 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.

```

933 \legacy_if:nF { measuring@ }
934 {
935   \str_set:Nx \l_tmpa_str { #1 }
936   \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
937   { \@@_error:nn { Duplicate~name } { #1 } }
938   { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
939   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
940 } ,
941 name .value_required:n = true ,
942 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
943 code-after .value_required:n = true ,
944 color-inside .code:n =
945   \bool_set_true:N \l_@@_color_inside_bool
946   \bool_set_true:N \l_@@_code_before_bool ,
947 color-inside .value_forbidden:n = true ,
948 colortbl-like .meta:n = color-inside
949 }
950 \keys_define:nn { NiceMatrix / notes }
951 {
952   para .bool_set:N = \l_@@_notes_para_bool ,
953   para .default:n = true ,
954   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
955   code-before .value_required:n = true ,
956   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
957   code-after .value_required:n = true ,
958   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
959   bottomrule .default:n = true ,
960   style .cs_set:Np = \@@_notes_style:n #1 ,
961   style .value_required:n = true ,
962   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
963   label-in-tabular .value_required:n = true ,
964   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
965   label-in-list .value_required:n = true ,
966   enumitem-keys .code:n =
967   {
968     \hook_gput_code:nnn { begindocument } { . }
969     {
970       \IfPackageLoadedTF { enumitem }
971       { \setlist* [ tabularnotes ] { #1 } }
972       { }
973     }
974   } ,
975   enumitem-keys .value_required:n = true ,
976   enumitem-keys-para .code:n =
977   {
978     \hook_gput_code:nnn { begindocument } { . }
979     {

```

```

980         \IfPackageLoadedTF { enumitem }
981         { \setlist* [ tabularnotes* ] { #1 } }
982         { }
983     }
984 },
985 enumitem-keys-para .value_required:n = true ,
986 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
987 detect-duplicates .default:n = true ,
988 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
989 }

990 \keys_define:nn { NiceMatrix / delimiters }
991 {
992     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
993     max-width .default:n = true ,
994     color .tl_set:N = \l_@@_delimiters_color_tl ,
995     color .value_required:n = true ,
996 }

```

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

```

997 \keys_define:nn { NiceMatrix }
998 {
999     NiceMatrixOptions .inherit:n =
1000     { NiceMatrix / Global } ,
1001     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1002     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1003     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1004     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1005     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1006     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1007     CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1008     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1009     NiceMatrix .inherit:n =
1010     {
1011         NiceMatrix / Global ,
1012         NiceMatrix / Env ,
1013     } ,
1014     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1015     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1016     NiceTabular .inherit:n =
1017     {
1018         NiceMatrix / Global ,
1019         NiceMatrix / Env
1020     } ,
1021     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1022     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1023     NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1024     NiceArray .inherit:n =
1025     {
1026         NiceMatrix / Global ,
1027         NiceMatrix / Env ,
1028     } ,
1029     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1030     NiceArray / rules .inherit:n = NiceMatrix / rules ,
1031     pNiceArray .inherit:n =
1032     {
1033         NiceMatrix / Global ,
1034         NiceMatrix / Env ,
1035     } ,
1036     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1037     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1038 }

```

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

```

1039 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1040 {
1041   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1042   delimiters / color .value_required:n = true ,
1043   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1044   delimiters / max-width .default:n = true ,
1045   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1046   delimiters .value_required:n = true ,
1047   width .dim_set:N = \l_@@_width_dim ,
1048   width .value_required:n = true ,
1049   last-col .code:n =
1050     \tl_if_empty:nF { #1 }
1051     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1052     \int_zero:N \l_@@_last_col_int ,
1053   small .bool_set:N = \l_@@_small_bool ,
1054   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.

```

1055   renew-matrix .code:n = \@@_renew_matrix: ,
1056   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}`.

```

1057   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.

```

1058   columns-width .code:n =
1059     \tl_if_eq:nnTF { #1 } { auto }
1060     { \@@_error:n { Option-auto-for-columns-width } }
1061     { \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.

```

1062   allow-duplicate-names .code:n =
1063     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1064   allow-duplicate-names .value_forbidden:n = true ,
1065   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1066   notes .value_required:n = true ,
1067   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1068   sub-matrix .value_required:n = true ,
1069   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1070   matrix / columns-type .value_required:n = true ,
1071   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1072   caption-above .default:n = true ,
1073   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1074 }

```

\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.

```

1075 \NewDocumentCommand \NiceMatrixOptions { m }
1076 { \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.

```

1077 \keys_define:nn { NiceMatrix / NiceMatrix }
1078 {
1079   last-col .code:n = \tl_if_empty:nTF { #1 }
1080     {
1081       \bool_set_true:N \l_@@_last_col_without_value_bool
1082       \int_set:Nn \l_@@_last_col_int { -1 }
1083     }
1084     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1085   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1086   columns-type .value_required:n = true ,
1087   l .meta:n = { columns-type = l } ,
1088   r .meta:n = { columns-type = r } ,
1089   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1090   delimiters / color .value_required:n = true ,
1091   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1092   delimiters / max-width .default:n = true ,
1093   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1094   delimiters .value_required:n = true ,
1095   small .bool_set:N = \l_@@_small_bool ,
1096   small .value_forbidden:n = true ,
1097   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1098 }

```

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

```

1099 \keys_define:nn { NiceMatrix / NiceArray }
1100 {

```

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.

```

1101   small .bool_set:N = \l_@@_small_bool ,
1102   small .value_forbidden:n = true ,
1103   last-col .code:n = \tl_if_empty:nF { #1 }
1104     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1105     { \int_zero:N \l_@@_last_col_int ,
1106   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1107   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1108   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1109 }

1110 \keys_define:nn { NiceMatrix / pNiceArray }
1111 {
1112   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1113   last-col .code:n = \tl_if_empty:nF { #1 }
1114     { \@@_error:n { last-col-non-empty-for-NiceArray } }
1115     { \int_zero:N \l_@@_last_col_int ,
1116   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1117   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1118   delimiters / color .value_required:n = true ,
1119   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1120   delimiters / max-width .default:n = true ,
1121   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1122   delimiters .value_required:n = true ,
1123   small .bool_set:N = \l_@@_small_bool ,
1124   small .value_forbidden:n = true ,
1125   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1126   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1127   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1128 }

```

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

```
1129 \keys_define:nn { NiceMatrix / NiceTabular }
1130 {
```

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.

```
1131 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1132 \bool_set_true:N \l_@@_width_used_bool ,
1133 width .value_required:n = true ,
1134 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1135 tabularnote .tl_set:N = \g_@@_tabularnote_tl ,
1136 tabularnote .value_required:n = true ,
1137 caption .tl_set:N = \l_@@_caption_tl ,
1138 caption .value_required:n = true ,
1139 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1140 short-caption .value_required:n = true ,
1141 label .tl_set:N = \l_@@_label_tl ,
1142 label .value_required:n = true ,
1143 last-col .code:n = \tl_if_empty:nF {#1}
1144 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1145 \int_zero:N \l_@@_last_col_int ,
1146 r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1147 l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1148 unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1149 }
```

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

```
1150 \keys_define:nn { NiceMatrix / CodeAfter }
1151 {
1152 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1153 delimiters / color .value_required:n = true ,
1154 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1155 rules .value_required:n = true ,
1156 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1157 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1158 sub-matrix .value_required:n = true ,
1159 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1160 }
```

## 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}).

```
1161 \cs_new_protected:Npn \@@_cell_begin:w
1162 {
```

\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.

```
1163 \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).

```
1164 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1165 \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.

```
1166 \int_compare:nNnT \c@jCol = \c_one_int
1167 { \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:`.

```
1168 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1169 \@@_tuning_not_tabular_begin:
```

```
1170 \@@_tuning_first_row:
```

```
1171 \@@_tuning_last_row:
```

```
1172 \g_@@_row_style_tl
```

```
1173 }
```

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

```
1174 \cs_new_protected:Npn \@@_tuning_first_row:
1175 {
1176   \int_if_zero:nT \c@iRow
1177   {
1178     \int_compare:nNnT \c@jCol > \c_zero_int
1179     {
1180       \l_@@_code_for_first_row_tl
1181       \xglobal \colorlet { nicematrix-first-row } { . }
1182     }
1183   }
1184 }
```

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

```
1185 \cs_new_protected:Npn \@@_tuning_last_row:
1186 {
1187   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1188   {
1189     \l_@@_code_for_last_row_tl
1190     \xglobal \colorlet { nicematrix-last-row } { . }
1191   }
1192 }
```

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

```
1193 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1194 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
```

```
1195 {
```

```
1196   \c_math_toggle_token
```

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

```
1197 \@@_tuning_key_small:
```

```
1198 }
```

```
1199 \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.

```

1200 \cs_new_protected:Npn \@@_begin_of_row:
1201 {
1202   \int_gincr:N \c@iRow
1203   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1204   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1205   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1206   \pgfpicture
1207   \pgfrememberpicturepositiononpagetrue
1208   \pgfcoordinate
1209   { \@@_env: - row - \int_use:N \c@iRow - base }
1210   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1211   \str_if_empty:NF \l_@@_name_str
1212   {
1213     \pgfnodealias
1214     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1215     { \@@_env: - row - \int_use:N \c@iRow - base }
1216   }
1217   \endpgfpicture
1218 }

```

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.

```

1219 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1220 {
1221   \int_if_zero:nTF \c@iRow
1222   {
1223     \dim_gset:Nn \g_@@_dp_row_zero_dim
1224     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1225     \dim_gset:Nn \g_@@_ht_row_zero_dim
1226     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1227   }
1228   {
1229     \int_compare:nNnT \c@iRow = \c_one_int
1230     {
1231       \dim_gset:Nn \g_@@_ht_row_one_dim
1232       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1233     }
1234   }
1235 }

1236 \cs_new_protected:Npn \@@_rotate_cell_box:
1237 {
1238   \box_rotate:Nn \l_@@_cell_box { 90 }
1239   \bool_if:NTF \g_@@_rotate_c_bool
1240   {
1241     \hbox_set:Nn \l_@@_cell_box
1242     {
1243       \c_math_toggle_token
1244       \vcenter { \box_use:N \l_@@_cell_box }
1245       \c_math_toggle_token
1246     }
1247   }
1248   {
1249     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1250     {
1251       \vbox_set_top:Nn \l_@@_cell_box
1252       {

```

```

1253         \vbox_to_zero:n { }
1254         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1255         \box_use:N \l_@@_cell_box
1256     }
1257 }
1258 }
1259 \bool_gset_false:N \g_@@_rotate_bool
1260 \bool_gset_false:N \g_@@_rotate_c_bool
1261 }
1262 \cs_new_protected:Npn \@@_adjust_size_box:
1263 {
1264     \dim_compare:nNt \g_@@_blocks_wd_dim > \c_zero_dim
1265     {
1266         \box_set_wd:Nn \l_@@_cell_box
1267         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1268         \dim_gzero:N \g_@@_blocks_wd_dim
1269     }
1270     \dim_compare:nNt \g_@@_blocks_dp_dim > \c_zero_dim
1271     {
1272         \box_set_dp:Nn \l_@@_cell_box
1273         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1274         \dim_gzero:N \g_@@_blocks_dp_dim
1275     }
1276     \dim_compare:nNt \g_@@_blocks_ht_dim > \c_zero_dim
1277     {
1278         \box_set_ht:Nn \l_@@_cell_box
1279         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1280         \dim_gzero:N \g_@@_blocks_ht_dim
1281     }
1282 }
1283 \cs_new_protected:Npn \@@_cell_end:
1284 {

```

The following command is nullified in the tabulars.

```

1285     \@@_tuning_not_tabular_end:
1286     \hbox_set_end:
1287     \@@_cell_end_i:
1288 }
1289 \cs_new_protected:Npn \@@_cell_end_i:
1290 {

```

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.

```

1291     \g_@@_cell_after_hook_tl
1292     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1293     \@@_adjust_size_box:
1294     \box_set_ht:Nn \l_@@_cell_box
1295     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1296     \box_set_dp:Nn \l_@@_cell_box
1297     { \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”).

```

1298     \@@_update_max_cell_width:

```

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

```

1299     \@@_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.

```

1300   \bool_if:NTF \g_@@_empty_cell_bool
1301   { \box_use_drop:N \l_@@_cell_box }
1302   {
1303     \bool_if:NTF \g_@@_not_empty_cell_bool
1304     \@@_node_for_cell:
1305     {
1306       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1307       \@@_node_for_cell:
1308       { \box_use_drop:N \l_@@_cell_box }
1309     }
1310   }
1311   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1312   \bool_gset_false:N \g_@@_empty_cell_bool
1313   \bool_gset_false:N \g_@@_not_empty_cell_bool
1314 }

```

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

```

1315 \cs_new_protected:Npn \@@_update_max_cell_width:
1316 {
1317   \dim_gset:Nn \g_@@_max_cell_width_dim
1318   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1319 }

```

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`).

```

1320 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1321 {
1322   \@@_math_toggle:
1323   \hbox_set_end:
1324   \bool_if:NF \g_@@_rotate_bool
1325   {
1326     \hbox_set:Nn \l_@@_cell_box
1327     {
1328       \makebox [ \l_@@_col_width_dim ] [ s ]
1329       { \hbox_unpack_drop:N \l_@@_cell_box }
1330     }
1331   }
1332   \@@_cell_end_i:
1333 }

```

```

1334 \pgfset
1335 {
1336   nicematrix / cell-node /.style =
1337   {
1338     inner-sep = \c_zero_dim ,
1339     minimum-width = \c_zero_dim

```

```

1340 }
1341 }

```

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

```

1342 \cs_new_protected:Npn \@@_node_for_cell:
1343 {
1344   \pgfpicture
1345   \pgfsetbaseline \c_zero_dim
1346   \pgfrememberpicturerepositiononpagetrue
1347   \pgfset { nicematrix / cell-node }
1348   \pgfnode
1349   { rectangle }
1350   { base }
1351   {

```

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).

```

1352     \set@color
1353     \box_use_drop:N \l_@@_cell_box
1354   }
1355   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1356   { \l_@@_pgf_node_code_tl }
1357   \str_if_empty:NF \l_@@_name_str
1358   {
1359     \pgfnodealias
1360     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1361     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1362   }
1363   \endpgfpicture
1364 }

```

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.

```

1365 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1366 {
1367   \cs_new_protected:Npn \@@_patch_node_for_cell:
1368   {
1369     \hbox_set:Nn \l_@@_cell_box
1370     {
1371       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1372       \hbox_overlap_left:n
1373       {
1374         \pgfsys@markposition
1375         { \@@_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.

```

1376         #1
1377       }
1378       \box_use:N \l_@@_cell_box
1379       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1380       \hbox_overlap_left:n
1381       {
1382         \pgfsys@markposition
1383         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1384       }
1385     }
1386   }
1387 }
1388 }

```

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

```

1389 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1390 {
1391   \@@_patch_node_for_cell:n
1392     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1393 }
1394 { \@@_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).

```

1395 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1396 {
1397   \bool_if:nTF { #1 } { \tl_gput_left:cx \tl_gput_right:cx
1398     { g_@@_ #2 _ lines _ tl }
1399     {
1400       \use:c { @@ _ draw _ #2 : nnn }
1401       { \int_use:N \c@iRow }
1402       { \int_use:N \c@jCol }
1403       { \exp_not:n { #3 } }
1404     }
1405 }

1406 \cs_new_protected:Npn \@@_array:
1407 {
1408   % \begin{macrocode}
1409   \dim_set:Nn \col@sep
1410     { \bool_if:nTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1411   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1412     { \cs_set_nopar:Npn \@halignto { } }
1413     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1414   \@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.

```

1415   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1416 }

```

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.

```

1417 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1418 \cs_new_protected:Npn \@@_create_row_node:
1419 {
1420   \int_compare:nNtT \c@iRow > \g_@@_last_row_node_int
1421   {
1422     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1423     \@@_create_row_node_i:
1424   }
1425 }

1426 \cs_new_protected:Npn \@@_create_row_node_i:
1427 {

```

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

```

1428   \hbox
1429   {
1430     \bool_if:NT \l_@@_code_before_bool
1431     {
1432       \vtop
1433       {
1434         \skip_vertical:N 0.5\arrayrulewidth
1435         \pgfsys@markposition
1436         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1437         \skip_vertical:N -0.5\arrayrulewidth
1438       }
1439     }
1440     \pgfpicture
1441     \pgfrememberpicturepositiononpagetrue
1442     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1443     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1444     \str_if_empty:NF \l_@@_name_str
1445     {
1446       \pgfnodealias
1447       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1448       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1449     }
1450     \endpgfpicture
1451   }
1452 }

```

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

```

1453 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1454 \cs_new_protected:Npn \@@_everycr_i:
1455 {
1456   \int_gzero:N \c@jCol
1457   \bool_gset_false:N \g_@@_after_col_zero_bool
1458   \bool_if:NF \g_@@_row_of_col_done_bool
1459   {
1460     \@@_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).

```

1461   \tl_if_empty:NF \l_@@_hlines_clist
1462   {
1463     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1464     {
1465       \exp_args:NNe
1466       \clist_if_in:NnT
1467       \l_@@_hlines_clist
1468       { \int_eval:n { \c@iRow + 1 } }
1469     }
1470   }

```

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.

```

1471         \int_compare:nNnT \c@iRow > { -1 }
1472         {
1473             \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@`.

```

1474             { \hrule height \arrayrulewidth width \c_zero_dim }
1475         }
1476     }
1477 }
1478 }
1479 }

```

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

```

1480 \cs_set_protected:Npn \@@_renew_dots:
1481 {
1482     \cs_set_eq:NN \ldots \@@_Ldots
1483     \cs_set_eq:NN \cdots \@@_Cdots
1484     \cs_set_eq:NN \vdots \@@_Vdots
1485     \cs_set_eq:NN \ddots \@@_Ddots
1486     \cs_set_eq:NN \iddots \@@_Iddots
1487     \cs_set_eq:NN \dots \@@_Ldots
1488     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1489 }
1490 \cs_new_protected:Npn \@@_test_color_inside:
1491 {
1492     \bool_if:NF \l_@@_color_inside_bool
1493     {

```

We will issue an error only during the first run.

```

1494         \bool_if:NF \g_@@_aux_found_bool
1495         { \@@_error:n { without~color~inside } }
1496     }
1497 }

```

```

1498 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1499 \hook_gput_code:nnn { begindocument } { . }
1500 {
1501     \IfPackageLoadedTF { colortbl }
1502     {
1503         \cs_set_protected:Npn \@@_redefine_everycr:
1504         {
1505             \CT@everycr
1506             {
1507                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1508                 \@@_everycr:
1509             }
1510         }
1511     }
1512     { }
1513 }

```

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

overwrite the previous definition of that row node and we have managed to avoid the error messages of that redefinition <sup>4</sup>.

```

1514 \hook_gput_code:nnn { begindocument } { . }
1515 {
1516   \IfPackageLoadedTF { booktabs }
1517   {
1518     \cs_new_protected:Npn \@@_patch_booktabs:
1519     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1520   }
1521   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1522 }

```

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

```

1523 \cs_new_protected:Npn \@@_pre_array_ii:
1524 {

```

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

```

1525   \int_gzero:N \g_@@_total_X_weight_int
1526   \@@_expand_clist:N \l_@@_hlines_clist
1527   \@@_expand_clist:N \l_@@_vlines_clist
1528   \@@_patch_booktabs:
1529   \box_clear_new:N \l_@@_cell_box
1530   \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}`).

```

1531   \bool_if:NT \l_@@_small_bool
1532   {
1533     \cs_set_nopar:Npn \arraystretch { 0.47 }
1534     \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_small_scriptstyle:` is null.

```

1535     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1536   }

```

```

1537   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1538   {
1539     \tl_put_right:Nn \@@_begin_of_row:
1540     {
1541       \pgfsys@markposition
1542       { \@@_env: - row - \int_use:N \c@iRow - base }
1543     }
1544   }

```

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`.

```

1545   \cs_set_nopar:Npn \ialign
1546   {
1547     \@@_redefine_everycr:
1548     \tabskip = \c_zero_skip

```

---

<sup>4</sup>cf. `\nicematrix@redefine@check@rerun`

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`<sup>5</sup> 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`.

```

1549      \dim_gzero_new:N \g_@@_dp_row_zero_dim
1550      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1551      \dim_gzero_new:N \g_@@_ht_row_zero_dim
1552      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1553      \dim_gzero_new:N \g_@@_ht_row_one_dim
1554      \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1555      \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1556      \dim_gzero_new:N \g_@@_ht_last_row_dim
1557      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1558      \dim_gzero_new:N \g_@@_dp_last_row_dim
1559      \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

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

```

1560      \cs_set_eq:NN \ialign \@_old_ialign:
1561      \halign
1562    }

```

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).

```

1563      \cs_set_eq:NN \@_old_ldots \ldots
1564      \cs_set_eq:NN \@_old_cdots \cdots
1565      \cs_set_eq:NN \@_old_vdots \vdots
1566      \cs_set_eq:NN \@_old_ddots \ddots
1567      \cs_set_eq:NN \@_old_iddots \iddots
1568      \bool_if:NTF \l_@@_standard_cline_bool
1569        { \cs_set_eq:NN \cline \@_standard_cline }
1570        { \cs_set_eq:NN \cline \@_cline }
1571      \cs_set_eq:NN \Ldots \@_Ldots
1572      \cs_set_eq:NN \Cdots \@_Cdots
1573      \cs_set_eq:NN \Vdots \@_Vdots
1574      \cs_set_eq:NN \Ddots \@_Ddots
1575      \cs_set_eq:NN \Iddots \@_Iddots
1576      \cs_set_eq:NN \Hline \@_Hline:
1577      \cs_set_eq:NN \Hspace \@_Hspace:
1578      \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1579      \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1580      \cs_set_eq:NN \Block \@_Block:
1581      \cs_set_eq:NN \rotate \@_rotate:
1582      \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1583      \cs_set_eq:NN \dotfill \@_dotfill:
1584      \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1585      \cs_set_eq:NN \diagbox \@_diagbox:nn
1586      \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1587      \cs_set_eq:NN \RowStyle \@_RowStyle:n
1588      \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1589        { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1590      \cs_set_eq:NN \cellcolor \@_cellcolor_tabular
1591      \cs_set_eq:NN \rowcolor \@_rowcolor_tabular
1592      \cs_set_eq:NN \rowcolors \@_rowcolors_tabular
1593      \cs_set_eq:NN \rowlistcolors \@_rowlistcolors_tabular
1594      \int_compare:nNt \l_@@_first_row_int > \c_zero_int

```

---

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

```

1595     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1596     \int_compare:nNt \l_@@_last_row_int < \c_zero_int
1597     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1598     \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.

```

1599     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1600     \hook_gput_code:nnn { env / tabular / begin } { . }
1601     { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1602     \@@_revert_colortbl:

```

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

```

1603     \tl_if_exist:NT \l_@@_note_in_caption_tl
1604     {
1605         \tl_if_empty:NF \l_@@_note_in_caption_tl
1606         {
1607             \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1608             \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1609         }
1610     }

```

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).

```

1611     \seq_gclear:N \g_@@_multicolumn_cells_seq
1612     \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).

```

1613     \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`).

```

1614     \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.

```

1615     \int_gzero_new:N \g_@@_col_total_int
1616     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1617     \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.

```

1618     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1619     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1620     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1621     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1622     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1623     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

```

```

1624     \tl_gclear:N \g_nicematrix_code_before_tl
1625     \tl_gclear:N \g_@@_pre_code_before_tl
1626 }

```

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

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

```

1627 \cs_new_protected:Npn \@@_pre_array:
1628 {
1629   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1630   \int_gzero_new:N \c@iRow
1631   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1632   \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).

```

1633   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1634   {
1635     \bool_set_true:N \l_@@_last_row_without_value_bool
1636     \bool_if:NT \g_@@_aux_found_bool
1637     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1638   }
1639   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1640   {
1641     \bool_if:NT \g_@@_aux_found_bool
1642     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1643   }

```

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”.

```

1644   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1645   {
1646     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1647     {
1648       \dim_gset:Nn \g_@@_ht_last_row_dim
1649       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1650       \dim_gset:Nn \g_@@_dp_last_row_dim
1651       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1652     }
1653   }

1654   \seq_gclear:N \g_@@_cols_vlism_seq
1655   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1656   \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.

```

1657   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the `aux` file.

```

1658   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1659   \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.

```
1660 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value  $-2$  is important.

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

```
1661 \@@_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.

```
1662 \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]`).

```
1663 \dim_zero_new:N \l_@@_left_delim_dim
1664 \dim_zero_new:N \l_@@_right_delim_dim
1665 \bool_if:NTF \g_@@_delims_bool
1666 {
```

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

```
1667 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1668 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1669 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1670 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1671 }
1672 {
1673 \dim_gset:Nn \l_@@_left_delim_dim
1674 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1675 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1676 }
```

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).

```
1677 \hbox_set:Nw \l_@@_the_array_box
1678 \skip_horizontal:N \l_@@_left_margin_dim
1679 \skip_horizontal:N \l_@@_extra_left_margin_dim
1680 \c_math_toggle_token
1681 \bool_if:NTF \l_@@_light_syntax_bool
1682 { \use:c { @@-light-syntax } }
1683 { \use:c { @@-normal-syntax } }
1684 }
```

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

```
1685 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1686 {
1687 \tl_set:Nn \l_tmpa_tl { #1 }
1688 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1689 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1690 \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1691 \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...

```
1692 \@@_pre_array:
1693 }
```

## 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).

```
1694 \cs_new_protected:Npn \@@_pre_code_before:
1695 {
```

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).

```
1696 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1697 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1698 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1699 \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.

```
1700 \pgfsys@markposition { \@@_env: - position }
1701 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1702 \pgfpicture
1703 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1704 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1705 {
1706 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1707 \pgfcoordinate { \@@_env: - row - ##1 }
1708 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1709 }
```

Now, the recreation of the `col` nodes.

```
1710 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1711 {
1712 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1713 \pgfcoordinate { \@@_env: - col - ##1 }
1714 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1715 }
```

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

```
1716 \@@_create_diag_nodes:
```

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

```
1717 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1718 \endpgfpicture
```

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

```
1719 \@@_create_blocks_nodes:
1720 \IfPackageLoadedTF { tikz }
1721 {
1722 \tikzset
1723 {
1724 every-picture / .style =
1725 { overlay , name-prefix = \@@_env: - }
1726 }
1727 }
1728 { }
1729 \cs_set_eq:NN \cellcolor \@@_cellcolor
1730 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1731 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1732 \cs_set_eq:NN \rowcolor \@@_rowcolor
```

```

1733 \cs_set_eq:NN \rowcolors \@@_rowcolors
1734 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1735 \cs_set_eq:NN \arraycolor \@@_arraycolor
1736 \cs_set_eq:NN \columncolor \@@_columncolor
1737 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1738 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1739 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1740 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1741 }

```

```

1742 \cs_new_protected:Npn \@@_exec_code_before:
1743 {
1744   \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`.

```

1745   \@@_add_to_colors_seq:nn { { nocolor } } { }
1746   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1747   \group_begin:

```

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

```

1748   \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`).

```

1749   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1750   { \@@_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`.

```

1751   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1752   \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.

```

1753   \@@_actually_color:
1754   \l_@@_code_before_tl
1755   \q_stop
1756   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1757   \group_end:
1758   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1759   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1760 }

```

```

1761 \keys_define:nn { NiceMatrix / CodeBefore }
1762 {
1763   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1764   create-cell-nodes .default:n = true ,
1765   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1766   sub-matrix .value_required:n = true ,
1767   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1768   delimiters / color .value_required:n = true ,
1769   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1770 }

```

```

1771 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1772 {
1773   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1774   \@@_CodeBefore:w
1775 }

```

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.

```

1776 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1777 {
1778   \bool_if:NT \g_@@_aux_found_bool
1779   {
1780     \@@_pre_code_before:
1781     #1
1782   }
1783 }

```

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.

```

1784 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1785 {
1786   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1787   {
1788     \pgfsys@getposition { \@@_env: - #1 - base } \@@_node_position:
1789     \pgfcoordinate { \@@_env: - row - #1 - base }
1790     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1791     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1792     {
1793       \cs_if_exist:cT
1794       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - #1 - #####1 - NW }
1795       {
1796         \pgfsys@getposition
1797         { \@@_env: - #1 - #####1 - NW }
1798         \@@_node_position:
1799         \pgfsys@getposition
1800         { \@@_env: - #1 - #####1 - SE }
1801         \@@_node_position_i:
1802         \@@_pgf_rect_node:nnn
1803         { \@@_env: - #1 - #####1 }
1804         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1805         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1806       }
1807     }
1808   }
1809   \int_step_inline:nn \c@iRow
1810   {
1811     \pgfnodealias
1812     { \@@_env: - #1 - last }
1813     { \@@_env: - #1 - \int_use:N \c@jCol }
1814   }
1815   \int_step_inline:nn \c@jCol
1816   {
1817     \pgfnodealias
1818     { \@@_env: - last - #1 }
1819     { \@@_env: - \int_use:N \c@iRow - #1 }
1820   }
1821   \@@_create_extra_nodes:
1822 }

```

```

1823 \cs_new_protected:Npn \@@_create_blocks_nodes:
1824 {
1825   \pgfpicture
1826   \pgf@relevantforpicturesizefalse
1827   \pgfrememberpicturepositiononpagetrue
1828   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1829   { \@@_create_one_block_node:nnnnn ##1 }
1830   \endpgfpicture
1831 }

```

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.<sup>6</sup>

```

1832 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1833 {
1834   \tl_if_empty:nF { #5 }
1835   {
1836     \@@_qpoint:n { col - #2 }
1837     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1838     \@@_qpoint:n { #1 }
1839     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1840     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1841     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1842     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1843     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1844     \@@_pgf_rect_node:nnnnn
1845     { \@@_env: - #5 }
1846     { \dim_use:N \l_tmpa_dim }
1847     { \dim_use:N \l_tmpb_dim }
1848     { \dim_use:N \l_@@_tmpc_dim }
1849     { \dim_use:N \l_@@_tmpd_dim }
1850   }
1851 }

```

```

1852 \cs_new_protected:Npn \@@_patch_for_revtext:
1853 {
1854   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1855   \cs_set_eq:NN \insert@column \insert@column@array
1856   \cs_set_eq:NN \@classx \@classx@array
1857   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1858   \cs_set_eq:NN \@arraycr \@arraycr@array
1859   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1860   \cs_set_eq:NN \array \array@array
1861   \cs_set_eq:NN \@array \@array@array
1862   \cs_set_eq:NN \@tabular \@tabular@array
1863   \cs_set_eq:NN \@mkpream \@mkpream@array
1864   \cs_set_eq:NN \endarray \endarray@array
1865   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1866   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1867 }

```

## 11 The environment `{NiceArrayWithDelims}`

```

1868 \NewDocumentEnvironment { NiceArrayWithDelims }
1869 { m m 0 { } m ! 0 { } t \CodeBefore }
1870 {

```

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

```

1871 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1872 \@@_provide_pgfsyspdfmark:
1873 \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.

```

1874 \bgroup

1875 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1876 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1877 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1878 \int_gzero:N \g_@@_block_box_int
1879 \dim_zero:N \g_@@_width_last_col_dim
1880 \dim_zero:N \g_@@_width_first_col_dim
1881 \bool_gset_false:N \g_@@_row_of_col_done_bool
1882 \str_if_empty:NT \g_@@_name_env_str
1883 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1884 \bool_if:NTF \l_@@_tabular_bool
1885 \mode_leave_vertical:
1886 \@@_test_if_math_mode:
1887 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1888 \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>7</sup>. 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.

```

1889 \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.

```

1890 \cs_if_exist:NT \tikz@library@external@loaded
1891 {
1892 \tikzexternaldisable
1893 \cs_if_exist:NT \ifstandalone
1894 { \tikzset { external / optimize = false } }
1895 }

```

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

```

1896 \int_gincr:N \g_@@_env_int
1897 \bool_if:NF \l_@@_block_auto_columns_width_bool
1898 { \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`).

```

1899 \seq_gclear:N \g_@@_blocks_seq
1900 \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`.

```

1901 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1902 \seq_gclear:N \g_@@_pos_of_xdots_seq
1903 \tl_gclear_new:N \g_@@_code_before_tl
1904 \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.

---

<sup>7</sup>e.g. `\color[rgb]{0.5,0.5,0}`

```

1905 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1906 {
1907     \bool_gset_true:N \g_@@_aux_found_bool
1908     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1909 }
1910 { \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.

```

1911 \tl_gclear:N \g_@@_aux_tl
1912 \tl_if_empty:NF \g_@@_code_before_tl
1913 {
1914     \bool_set_true:N \l_@@_code_before_bool
1915     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1916 }
1917 \tl_if_empty:NF \g_@@_pre_code_before_tl
1918 { \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.

```

1919 \bool_if:NTF \g_@@_delims_bool
1920 { \keys_set:nn { NiceMatrix / pNiceArray } }
1921 { \keys_set:nn { NiceMatrix / NiceArray } }
1922 { #3 , #5 }

```

```

1923 \@@_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:.

```

1924 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1925 }

```

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

```

1926 {
1927     \bool_if:NTF \l_@@_light_syntax_bool
1928     { \use:c { end @@-light-syntax } }
1929     { \use:c { end @@-normal-syntax } }
1930     \c_math_toggle_token
1931     \skip_horizontal:N \l_@@_right_margin_dim
1932     \skip_horizontal:N \l_@@_extra_right_margin_dim
1933     \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.

```

1934 \bool_if:NT \l_@@_width_used_bool
1935 {
1936     \int_if_zero:nT \g_@@_total_X_weight_int
1937     { \@@_error_or_warning:n { width-without-X-columns } }
1938 }

```

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.

```

1939 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1940 {
1941     \tl_gput_right:Nx \g_@@_aux_tl
1942     {

```

```

1943 \bool_set_true:N \l_@@_X_columns_aux_bool
1944 \dim_set:Nn \l_@@_X_columns_dim
1945 {
1946   \dim_compare:nNnTF
1947   {
1948     \dim_abs:n
1949     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1950   }
1951   <
1952   { 0.001 pt }
1953   { \dim_use:N \l_@@_X_columns_dim }
1954   {
1955     \dim_eval:n
1956     {
1957       ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1958       / \int_use:N \g_@@_total_X_weight_int
1959       + \l_@@_X_columns_dim
1960     }
1961   }
1962 }
1963 }
1964 }

```

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

```

1965 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1966 {
1967   \bool_if:NF \l_@@_last_row_without_value_bool
1968   {
1969     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1970     {
1971       \@@_error:n { Wrong-last-row }
1972       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1973     }
1974   }
1975 }

```

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”.<sup>8</sup>

```

1976 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1977 \bool_if:NnTF \g_@@_last_col_found_bool
1978 { \int_gdecr:N \c@jCol }
1979 {
1980   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1981   { \@@_error:n { last-col-not-used } }
1982 }

```

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

```

1983 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1984 \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).

```

1985 \int_if_zero:nT \l_@@_first_col_int
1986 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

1987 \bool_if:nTF { ! \g_@@_delims_bool }
1988 {

```

---

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

```

1989 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
1990 \@@_use_arraybox_with_notes_c:
1991 {
1992 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
1993 \@@_use_arraybox_with_notes_b:
1994 \@@_use_arraybox_with_notes:
1995 }
1996 }

```

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).

```

1997 {
1998 \int_if_zero:nTF \l_@@_first_row_int
1999 {
2000 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2001 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2002 }
2003 { \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”.<sup>9</sup>

```

2004 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2005 {
2006 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2007 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2008 }
2009 { \dim_zero:N \l_tmpb_dim }
2010 \hbox_set:Nn \l_tmpa_box
2011 {
2012 \c_math_toggle_token
2013 \@@_color:o \l_@@_delimiters_color_tl
2014 \exp_after:wN \left \g_@@_left_delim_tl
2015 \vcenter
2016 {

```

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.

```

2017 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2018 \hbox
2019 {
2020 \bool_if:NTF \l_@@_tabular_bool
2021 { \skip_horizontal:N -\tabcolsep }
2022 { \skip_horizontal:N -\arraycolsep }
2023 \@@_use_arraybox_with_notes_c:
2024 \bool_if:NTF \l_@@_tabular_bool
2025 { \skip_horizontal:N -\tabcolsep }
2026 { \skip_horizontal:N -\arraycolsep }
2027 }

```

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

```

2028 \skip_vertical:N -\l_tmpb_dim
2029 \skip_vertical:N \arrayrulewidth
2030 }
2031 \exp_after:wN \right \g_@@_right_delim_tl
2032 \c_math_toggle_token
2033 }

```

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.

```

2034 \bool_if:NTF \l_@@_delimiters_max_width_bool
2035 {

```

---

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

```

2036         \@@_put_box_in_flow_bis:nn
2037         \g_@@_left_delim_tl
2038         \g_@@_right_delim_tl
2039     }
2040     \@@_put_box_in_flow:
2041 }

```

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).

```

2042 \bool_if:NT \g_@@_last_col_found_bool
2043 { \skip_horizontal:N \g_@@_width_last_col_dim }
2044 \bool_if:NT \l_@@_preamble_bool
2045 {
2046     \int_compare:nNtT \c@jCol < \g_@@_static_num_of_col_int
2047     { \@@_warning_gredirect_none:n { columns~not~used } }
2048 }
2049 \@@_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.

```

2050 \egroup

```

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

```

2051 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2052 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2053 \iow_now:Nx \@mainaux
2054 {
2055     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2056     { \exp_not:o \g_@@_aux_tl }
2057 }
2058 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2059 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2060 }

```

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.

```

2061 \cs_new_protected:Npn \@@_transform_preamble:
2062 {
2063     \@@_transform_preamble_i:
2064     \@@_transform_preamble_ii:
2065 }
2066 \cs_new_protected:Npn \@@_transform_preamble_i:
2067 {
2068     \int_gzero:N \c@jCol

```

The sequence `\g_@@_cols_vlism_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`).

```

2069 \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.

```

2070 \bool_gset_false:N \g_tmpb_bool

```

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

```
2071 \tl_gclear_new:N \g_@@_pre_cell_tl
```

The counter \l\_tmpa\_int will count the number of consecutive occurrences of the symbol |.

```
2072 \int_zero:N \l_tmpa_int
2073 \tl_gclear:N \g_@@_array_preamble_tl
2074 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2075 {
2076   \tl_gset:Nn \g_@@_array_preamble_tl
2077   { ! { \skip_horizontal:N \arrayrulewidth } }
2078 }
2079 {
2080   \clist_if_in:NnT \l_@@_vlines_clist 1
2081   {
2082     \tl_gset:Nn \g_@@_array_preamble_tl
2083     { ! { \skip_horizontal:N \arrayrulewidth } }
2084   }
2085 }
```

Now, we actually make the preamble (which will be given to {array}). It will be stored in \g\_@@\_array\_preamble\_tl.

```
2086 \exp_last_unbraced:N \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2087 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2088 \@@_replace_columncolor:
2089 }
```

```
2090 \hook_gput_code:nnn { begindocument } { . }
2091 {
2092   \IfPackageLoadedTF { colortbl }
2093   {
2094     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2095     \cs_new_protected:Npn \@@_replace_columncolor:
2096     {
2097       \regex_replace_all:NnN
2098       \c_@@_columncolor_regex
2099       { \c { @@_columncolor_preamble } }
2100       \g_@@_array_preamble_tl
2101     }
2102   }
2103   {
2104     \cs_new_protected:Npn \@@_replace_columncolor:
2105     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2106   }
2107 }
```

```
2108 \cs_new_protected:Npn \@@_transform_preamble_ii:
2109 {
```

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

```
2110 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2111 {
2112   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2113   { \bool_gset_true:N \g_@@_delims_bool }
2114 }
2115 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2116 \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).

```

2117 \int_if_zero:nTF \l_@@_first_col_int
2118 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2119 {
2120   \bool_if:NF \g_@@_delims_bool
2121   {
2122     \bool_if:NF \l_@@_tabular_bool
2123     {
2124       \tl_if_empty:NT \l_@@_vlines_clist
2125       {
2126         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2127         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2128       }
2129     }
2130   }
2131 }
2132 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2133 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2134 {
2135   \bool_if:NF \g_@@_delims_bool
2136   {
2137     \bool_if:NF \l_@@_tabular_bool
2138     {
2139       \tl_if_empty:NT \l_@@_vlines_clist
2140       {
2141         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2142         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2143       }
2144     }
2145   }
2146 }

```

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`).

```

2147 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2148 {
2149   \tl_gput_right:Nn \g_@@_array_preamble_tl
2150   { > { \@@_error_too_much_cols: } 1 }
2151 }
2152 }

```

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.

```

2153 \cs_new_protected:Npn \@@_rec_preamble:n #1
2154 {

```

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.<sup>10</sup>

```

2155 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2156 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2157 {

```

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

```

2158 \cs_if_exist:cTF { NC @ find @ #1 }
2159 {
2160   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }

```

---

<sup>10</sup>We 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`.

```

2161         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2162     }
2163     {
2164         \tl_if_eq:nnT { #1 } { S }
2165         { \@@_fatal:n { unknown~column~type~S } }
2166         { \@@_fatal:nn { unknown~column~type } { #1 } }
2167     }
2168 }
2169 }

```

For c, l and r

```

2170 \cs_new:Npn \@@_c #1
2171 {
2172     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2173     \tl_gclear:N \g_@@_pre_cell_tl
2174     \tl_gput_right:Nn \g_@@_array_preamble_tl
2175     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2176     \int_gincr:N \c@jCol
2177     \@@_rec_preamble_after_col:n
2178 }

2179 \cs_new:Npn \@@_l #1
2180 {
2181     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2182     \tl_gclear:N \g_@@_pre_cell_tl
2183     \tl_gput_right:Nn \g_@@_array_preamble_tl
2184     {
2185         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2186         l
2187         < \@@_cell_end:
2188     }
2189     \int_gincr:N \c@jCol
2190     \@@_rec_preamble_after_col:n
2191 }

2192 \cs_new:Npn \@@_r #1
2193 {
2194     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2195     \tl_gclear:N \g_@@_pre_cell_tl
2196     \tl_gput_right:Nn \g_@@_array_preamble_tl
2197     {
2198         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2199         r
2200         < \@@_cell_end:
2201     }
2202     \int_gincr:N \c@jCol
2203     \@@_rec_preamble_after_col:n
2204 }

```

For ! and @

```

2205 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2206 {
2207     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2208     \@@_rec_preamble:n
2209 }
2210 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2211 \cs_new:cpn { @@ _ | } #1
2212 {

```

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

```

2213 \int_incr:N \l_tmpa_int
2214 \@@_make_preamble_i_i:n
2215 }

2216 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2217 {
2218   \str_if_eq:nnTF { #1 } |
2219   { \use:c { @@ _ | } | }
2220   { \@@_make_preamble_i_ii:nn { } #1 }
2221 }

2222 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2223 {
2224   \str_if_eq:nnTF { #2 } [
2225   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2226   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2227 }

2228 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2229 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2230 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2231 {
2232   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2233   \tl_gput_right:Nx \g_@@_array_preamble_tl
2234   {

```

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

```

2235   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2236   }
2237   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2238   {
2239     \@@_vline:n
2240     {
2241       position = \int_eval:n { \c@jCol + 1 } ,
2242       multiplicity = \int_use:N \l_tmpa_int ,
2243       total-width = \dim_use:N \l_@@_rule_width_dim ,
2244       #2
2245     }

```

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.

```

2246   }
2247   \int_zero:N \l_tmpa_int
2248   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2249   \@@_rec_preamble:n #1
2250 }

2251 \cs_new:cpn { @@ _ > } #1 #2
2252 {
2253   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2254   \@@_rec_preamble:n
2255 }

2256 \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.

```

2257 \keys_define:nn { WithArrows / p-column }
2258 {
2259   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2260   r .value_forbidden:n = true ,
2261   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2262   c .value_forbidden:n = true ,

```

```

2263 l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2264 l .value_forbidden:n = true ,
2265 R .code:n =
2266 \IfPackageLoadedTF { ragged2e }
2267 { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2268 {
2269 \@@_error_or_warning:n { ragged2e~not~loaded }
2270 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2271 } ,
2272 R .value_forbidden:n = true ,
2273 L .code:n =
2274 \IfPackageLoadedTF { ragged2e }
2275 { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2276 {
2277 \@@_error_or_warning:n { ragged2e~not~loaded }
2278 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2279 } ,
2280 L .value_forbidden:n = true ,
2281 C .code:n =
2282 \IfPackageLoadedTF { ragged2e }
2283 { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2284 {
2285 \@@_error_or_warning:n { ragged2e~not~loaded }
2286 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2287 } ,
2288 C .value_forbidden:n = true ,
2289 S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2290 S .value_forbidden:n = true ,
2291 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2292 p .value_forbidden:n = true ,
2293 t .meta:n = p ,
2294 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2295 m .value_forbidden:n = true ,
2296 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2297 b .value_forbidden:n = true ,
2298 }

```

For p, b and m.

```

2299 \cs_new:Npn \@@_p #1
2300 {
2301 \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2302 \@@_make_preamble_ii_i:n
2303 }
2304 \cs_set_eq:NN \@@_b \@@_p
2305 \cs_set_eq:NN \@@_m \@@_p
2306 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2307 {
2308 \str_if_eq:nnTF { #1 } { [ ]
2309 { \@@_make_preamble_ii_ii:w [ ]
2310 { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2311 }
2312 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2313 { \@@_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.

```

2314 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2315 {

```

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).

```

2316 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2317 \@@_keys_p_column:n { #1 }
2318 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2319 }

2320 \cs_new_protected:Npn \@@_keys_p_column:n #1
2321 { \keys_set_known:nnN { WithArrows / 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.

```

2322 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2323 {
2324   \use:e
2325   {
2326     \@@_make_preamble_ii_v:nnnnnnnn
2327     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2328     { \dim_eval:n { #1 } }
2329     {

```

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.

```

2330 \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2331 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2332 {
2333   \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2334   { \str_lowercase:V \l_@@_hpos_col_str }
2335 }
2336 \str_case:on \l_@@_hpos_col_str
2337 {
2338   c { \exp_not:N \centering }
2339   l { \exp_not:N \raggedright }
2340   r { \exp_not:N \raggedleft }
2341   C { \exp_not:N \Centering }
2342   L { \exp_not:N \RaggedRight }
2343   R { \exp_not:N \RaggedLeft }
2344 }
2345 #3
2346 }
2347 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2348 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2349 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2350 { #2 }
2351 {
2352   \str_case:onF \l_@@_hpos_col_str
2353   {
2354     { j } { c }
2355     { si } { c }
2356   }

```

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

```

2357 { \str_lowercase:V \l_@@_hpos_col_str }
2358 }
2359 }

```

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

```

2360 \int_gincr:N \c@jCol
2361 \@@_rec_preamble_after_col:n
2362 }

```

#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*.

```

2363 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2364 {
2365   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2366     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2367     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2368   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2369   \tl_gc_clear:N \g_@@_pre_cell_tl
2370   \tl_gput_right:Nn \g_@@_array_preamble_tl
2371     {
2372       > {

```

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.

```

2373       \dim_set:Nn \l_@@_col_width_dim { #2 }
2374       \@@_cell_begin:w

```

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

```

2375       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2376       \everypar
2377       {
2378         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2379         \everypar { }
2380       }

```

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

```

2381       #3

```

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

```

2382       \g_@@_row_style_tl
2383       \arraybackslash
2384       #5
2385     }
2386     #8
2387     < {
2388       #6

```

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

```

2389       \@finalstrut \@arstrutbox
2390       \use:c { end #7 }

```

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

```

2391       #4
2392       \@@_cell_end:
2393     }
2394   }
2395 }

```

```

2396 \str_new:N \c_@@_ignorespaces_str
2397 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2398 \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...

```

2399 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2400 \cs_new_protected:Npn \@@_test_if_empty_i:
2401 {
2402   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2403   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2404   { \@@_test_if_empty:w }
2405 }
2406 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2407 {
2408   \peek_meaning:NT \unskip
2409   {
2410     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2411     {
2412       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2413       \skip_horizontal:N \l_@@_col_width_dim
2414     }
2415   }
2416 }
2417 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2418 {
2419   \peek_meaning:NT \__siunitx_table_skip:n
2420   {
2421     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2422     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2423   }
2424 }

```

The following command will be used in `m-columns` in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more that the height of `\@arstrutbox`, there is only one row.

```

2425 \cs_new_protected:Npn \@@_center_cell_box:
2426 {

```

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

```

2427   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2428   {
2429     \int_compare:nNnT
2430     { \box_ht:N \l_@@_cell_box }
2431     >

```

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).

```

2432     { \box_ht:N \strutbox }
2433     {
2434       \hbox_set:Nn \l_@@_cell_box
2435       {
2436         \box_move_down:nn
2437         {
2438           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2439             + \baselineskip ) / 2
2440         }
2441         { \box_use:N \l_@@_cell_box }
2442     }
2443 }
2444 }
2445 }

```

For V (similar to the V of varwidth).

```

2446 \cs_new:Npn \@@_V #1 #2
2447 {
2448     \str_if_eq:nnTF { #2 } { [ ] }
2449     { \@@_make_preamble_V_i:w [ ] }
2450     { \@@_make_preamble_V_i:w [ ] { #2 } }
2451 }
2452 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2453 { \@@_make_preamble_V_ii:nn { #1 } }
2454 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2455 {
2456     \str_set:Nn \l_@@_vpos_col_str { p }
2457     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2458     \@@_keys_p_column:n { #1 }
2459     \IfPackageLoadedTF { varwidth }
2460     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2461     {
2462         \@@_error_or_warning:n { varwidth-not-loaded }
2463         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2464     }
2465 }

```

For w and W

```

2466 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2467 \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.

```

2468 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2469 {
2470     \str_if_eq:nnTF { #3 } { s }
2471     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2472     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2473 }

```

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.

```

2474 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2475 {
2476     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2477     \tl_gclear:N \g_@@_pre_cell_tl
2478     \tl_gput_right:Nn \g_@@_array_preamble_tl
2479     {
2480         > {
2481             \dim_set:Nn \l_@@_col_width_dim { #2 }
2482             \@@_cell_begin:w
2483             \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2484         }
2485         c
2486         < {
2487             \@@_cell_end_for_w_s:

```

```

2488         #1
2489         \@@_adjust_size_box:
2490         \box_use_drop:N \l_@@_cell_box
2491     }
2492 }
2493 \int_gincr:N \c@jCol
2494 \@@_rec_preamble_after_col:n
2495 }

```

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

```

2496 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2497 {
2498     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2499     \tl_gclear:N \g_@@_pre_cell_tl
2500     \tl_gput_right:Nn \g_@@_array_preamble_tl
2501     {
2502         > {

```

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.

```

2503         \dim_set:Nn \l_@@_col_width_dim { #4 }
2504         \hbox_set:Nw \l_@@_cell_box
2505         \@@_cell_begin:w
2506         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2507     }
2508     c
2509     < {
2510         \@@_cell_end:
2511         \hbox_set_end:
2512         #1
2513         \@@_adjust_size_box:
2514         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2515     }
2516 }

```

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

```

2517     \int_gincr:N \c@jCol
2518     \@@_rec_preamble_after_col:n
2519 }

```

```

2520 \cs_new_protected:Npn \@@_special_W:
2521 {
2522     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2523     { \@@_warning:n { W~warning } }
2524 }

```

For S (of siunitx).

```

2525 \cs_new:Npn \@@_S #1 #2
2526 {
2527     \str_if_eq:nnTF { #2 } { [ ] }
2528     { \@@_make_preamble_S:w [ ] }
2529     { \@@_make_preamble_S:w [ ] { #2 } }
2530 }
2531 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2532 { \@@_make_preamble_S:i:n { #1 } }
2533 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2534 {
2535     \IfPackageLoadedTF { siunitx }
2536     {
2537         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2538         \tl_gclear:N \g_@@_pre_cell_tl
2539         \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2540     {
2541     > {
2542         \@@_cell_begin:w
2543         \keys_set:nn { siunitx } { #1 }
2544         \siunitx_cell_begin:w
2545     }
2546     c
2547     < { \siunitx_cell_end: \@@_cell_end: }
2548 }

```

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

```

2549     \int_gincr:N \c@jCol
2550     \@@_rec_preamble_after_col:n
2551 }
2552 { \@@_fatal:n { siunitx~not~loaded } }
2553 }

```

For (, [ and \{.

```

2554 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2555 {
2556     \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.

```

2557     \int_if_zero:nTF \c@jCol
2558     {
2559         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2560         {

```

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

```

2561         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2562         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2563         \@@_rec_preamble:n #2
2564     }
2565     {
2566         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2567         \@@_make_preamble_iv:nn { #1 } { #2 }
2568     }
2569 }
2570 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2571 }
2572 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2573 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2574 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2575 {
2576     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2577     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2578     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2579     {
2580         \@@_error:nn { delimiter~after~opening } { #2 }
2581         \@@_rec_preamble:n
2582     }
2583     { \@@_rec_preamble:n #2 }
2584 }

```

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

```

2585 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2586 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2

```

```

2587 {
2588   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2589   \tl_if_in:nnTF { } ] \} { #2 }
2590   { \@@_make_preamble_v:nnn #1 #2 }
2591   {
2592     \tl_if_eq:nnTF { \stop } { #2 }
2593     {
2594       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2595       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2596       {
2597         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2598         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2599         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2600         \@@_rec_preamble:n #2
2601       }
2602     }
2603     {
2604       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2605       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2606       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2607       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2608       \@@_rec_preamble:n #2
2609     }
2610   }
2611 }
2612 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2613 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2614 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2615 {
2616   \tl_if_eq:nnTF { \stop } { #3 }
2617   {
2618     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2619     {
2620       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2621       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2622       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2623       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2624     }
2625     {
2626       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2627       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2628       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2629       \@@_error:nn { double~closing~delimiter } { #2 }
2630     }
2631   }
2632   {
2633     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2634     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2635     \@@_error:nn { double~closing~delimiter } { #2 }
2636     \@@_rec_preamble:n #3
2637   }
2638 }
2639 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2640 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2641 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2642 {
2643   \str_if_eq:nnTF { #1 } { < }

```

```

2644 \@@_rec_preamble_after_col_i:n
2645 {
2646   \str_if_eq:nnTF { #1 } { @ }
2647   \@@_rec_preamble_after_col_ii:n
2648   {
2649     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2650     {
2651       \tl_gput_right:Nn \g_@@_array_preamble_tl
2652       { ! { \skip_horizontal:N \arrayrulewidth } }
2653     }
2654     {
2655       \exp_args:NNe
2656       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2657       {
2658         \tl_gput_right:Nn \g_@@_array_preamble_tl
2659         { ! { \skip_horizontal:N \arrayrulewidth } }
2660       }
2661     }
2662     \@@_rec_preamble:n { #1 }
2663   }
2664 }
2665 }
2666 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2667 {
2668   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2669   \@@_rec_preamble_after_col:n
2670 }

```

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

```

2671 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2672 {
2673   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2674   {
2675     \tl_gput_right:Nn \g_@@_array_preamble_tl
2676     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2677   }
2678   {
2679     \exp_args:NNe
2680     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2681     {
2682       \tl_gput_right:Nn \g_@@_array_preamble_tl
2683       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2684     }
2685     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2686   }
2687   \@@_rec_preamble:n
2688 }
2689 \cs_new:cpn { @@ _ * } #1 #2 #3
2690 {
2691   \tl_clear:N \l_tmpa_tl
2692   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2693   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2694 }

```

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

```

2695 \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.

```

2696 \cs_new:Npn \@@_X #1 #2
2697 {
2698   \str_if_eq:nnTF { #2 } { [ ]
2699     { \@@_make_preamble_X:w [ ]
2700       { \@@_make_preamble_X:w [ ] #2 }
2701     }
2702   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2703     { \@@_make_preamble_X:i:n { #1 } }

```

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

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

```

2704 \keys_define:nn { WithArrows / X-column }
2705 { 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.

```

2706 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2707 {

```

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).

```

2708   \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).

```

2709   \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.

```

2710   \int_zero_new:N \l_@@_weight_int
2711   \int_set_eq:NN \l_@@_weight_int \c_one_int
2712   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l\_tmpa\_tl

```

2713   \keys_set:no { WithArrows / X-column } \l_tmpa_tl
2714   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2715     {
2716       \@@_error_or_warning:n { negative-weight }
2717       \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2718     }
2719   \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).

```

2720   \bool_if:NTF \l_@@_X_columns_aux_bool
2721     {
2722       \exp_args:Nne
2723       \@@_make_preamble_ii_iv:nnn
2724         { \l_@@_weight_int \l_@@_X_columns_dim }
2725         { minipage }
2726         { \@@_no_update_width: }
2727     }
2728     {
2729       \tl_gput_right:Nn \g_@@_array_preamble_tl
2730         {
2731           > {
2732             \@@_cell_begin:w
2733             \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`.

```
2734 \NotEmpty
```

The following code will nullify the box of the cell.

```
2735 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2736 { \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`.

```
2737 \begin { minipage } { 5 cm } \arraybackslash
2738 }
2739 c
2740 < {
2741 \end { minipage }
2742 \@@_cell_end:
2743 }
2744 }
2745 \int_gincr:N \c@jCol
2746 \@@_rec_preamble_after_col:n
2747 }
2748 }
```

```
2749 \cs_new_protected:Npn \@@_no_update_width:
2750 {
2751 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2752 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2753 }
```

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

```
2754 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2755 {
2756 \seq_gput_right:Nx \g_@@_cols_vlism_seq
2757 { \int_eval:n { \c@jCol + 1 } }
2758 \tl_gput_right:Nx \g_@@_array_preamble_tl
2759 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2760 \@@_rec_preamble:n
2761 }
```

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.

```
2762 \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).

```
2763 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2764 { \@@_fatal:n { Preamble~forgotten } }
2765 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2766 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2767 \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).

```
2768 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2769 {
```

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`.

```

2770 \multispan { #1 }
2771 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2772 \begingroup
2773 \cs_set:Npn \@addamp
2774 { \legacy_if:nTF { \@firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2775 \tl_gclear:N \g_@@_preamble_tl
2776 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2777 \exp_args:No \@mkpream \g_@@_preamble_tl
2778 \@addtopreamble \@empty
2779 \endgroup

```

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

```

2780 \int_compare:nNnT { #1 } > \c_one_int
2781 {
2782   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2783   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2784   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2785   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2786   {
2787     {
2788       \int_if_zero:nTF \c@jCol
2789       { \int_eval:n { \c@iRow + 1 } }
2790       { \int_use:N \c@iRow }
2791     }
2792     { \int_eval:n { \c@jCol + 1 } }
2793     {
2794       \int_if_zero:nTF \c@jCol
2795       { \int_eval:n { \c@iRow + 1 } }
2796       { \int_use:N \c@iRow }
2797     }
2798     { \int_eval:n { \c@jCol + #1 } }
2799     { } % for the name of the block
2800   }
2801 }

```

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

```

2802 \cs_set:Npn \@sharp { #3 }
2803 \@arstrut
2804 \@preamble
2805 \null

```

We add some lines.

```

2806 \int_gadd:Nn \c@jCol { #1 - 1 }
2807 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2808 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2809 \ignorespaces
2810 }

```

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`.

```

2811 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2812 {
2813   \str_case:nnF { #1 }
2814   {

```

```

2815 c { \@@_make_m_preamble_i:n #1 }
2816 l { \@@_make_m_preamble_i:n #1 }
2817 r { \@@_make_m_preamble_i:n #1 }
2818 > { \@@_make_m_preamble_ii:nn #1 }
2819 ! { \@@_make_m_preamble_ii:nn #1 }
2820 @ { \@@_make_m_preamble_ii:nn #1 }
2821 | { \@@_make_m_preamble_iii:n #1 }
2822 p { \@@_make_m_preamble_iv:nnn t #1 }
2823 m { \@@_make_m_preamble_iv:nnn c #1 }
2824 b { \@@_make_m_preamble_iv:nnn b #1 }
2825 w { \@@_make_m_preamble_v:nnnn { } #1 }
2826 W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2827 \q_stop { }
2828 }
2829 {
2830 \cs_if_exist:cTF { NC @ find @ #1 }
2831 {
2832 \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2833 \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2834 }
2835 {
2836 \tl_if_eq:nnT { #1 } { S }
2837 { \@@_fatal:n { unknown~column~type~S } }
2838 { \@@_fatal:nn { unknown~column~type } { #1 } }
2839 }
2840 }
2841 }

```

For c, l and r

```

2842 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2843 {
2844 \tl_gput_right:Nn \g_@@_preamble_tl
2845 {
2846 > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2847 #1
2848 < \@@_cell_end:
2849 }

```

We test for the presence of a <.

```

2850 \@@_make_m_preamble_x:n
2851 }

```

For >, ! and @

```

2852 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2853 {
2854 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2855 \@@_make_m_preamble:n
2856 }

```

For |

```

2857 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2858 {
2859 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2860 \@@_make_m_preamble:n
2861 }

```

For p, m and b

```

2862 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2863 {
2864 \tl_gput_right:Nn \g_@@_preamble_tl
2865 {
2866 > {
2867 \@@_cell_begin:w

```

```

2868         \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2869         \mode_leave_vertical:
2870         \arraybackslash
2871         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2872     }
2873     c
2874     < {
2875         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2876         \end { minipage }
2877         \@@_cell_end:
2878     }
2879 }

```

We test for the presence of a <.

```

2880     \@@_make_m_preamble_x:n
2881 }

```

For w and W

```

2882 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2883 {
2884     \tl_gput_right:Nn \g_@@_preamble_tl
2885     {
2886         > {
2887             \dim_set:Nn \l_@@_col_width_dim { #4 }
2888             \hbox_set:Nw \l_@@_cell_box
2889             \@@_cell_begin:w
2890             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2891         }
2892         c
2893         < {
2894             \@@_cell_end:
2895             \hbox_set_end:
2896             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2897             #1
2898             \@@_adjust_size_box:
2899             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2900         }
2901     }

```

We test for the presence of a <.

```

2902     \@@_make_m_preamble_x:n
2903 }

```

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

```

2904 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2905 {
2906     \str_if_eq:nnTF { #1 } { < }
2907     \@@_make_m_preamble_ix:n
2908     { \@@_make_m_preamble:n { #1 } }
2909 }
2910 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2911 {
2912     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2913     \@@_make_m_preamble_x:n
2914 }

```

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).

```

2915 \cs_new_protected:Npn \@@_put_box_in_flow:
2916 {
2917     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }

```

```

2918 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2919 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2920 { \box_use_drop:N \l_tmpa_box }
2921 \@@_put_box_in_flow_i:
2922 }

```

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).

```

2923 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2924 {
2925   \pgfpicture
2926     \@@_qpoint:n { row - 1 }
2927     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2928     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2929     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2930     \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).

```

2931 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2932 {
2933   \int_set:Nn \l_tmpa_int
2934   {
2935     \str_range:Nnn
2936     \l_@@_baseline_tl
2937     6
2938     { \tl_count:o \l_@@_baseline_tl }
2939   }
2940   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2941 }
2942 {
2943   \tl_if_eq:NnTF \l_@@_baseline_tl { t }
2944   { \int_set_eq:NN \l_tmpa_int \c_one_int }
2945   {
2946     \tl_if_eq:NnTF \l_@@_baseline_tl { b }
2947     { \int_set_eq:NN \l_tmpa_int \c@iRow }
2948     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2949   }
2950   \bool_lazy_or:nnT
2951   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2952   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2953   {
2954     \@@_error:n { bad-value-for-baseline }
2955     \int_set_eq:NN \l_tmpa_int \c_one_int
2956   }
2957   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2958 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2959 }
2960 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2961 \endpgfpicture
2962 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2963 \box_use_drop:N \l_tmpa_box
2964 }

```

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).

```

2965 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2966 {

```

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.

```

2967 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2968 {
2969   \int_compare:nNnT \c@jCol > \c_one_int
2970   {
2971     \box_set_wd:Nn \l_@@_the_array_box
2972     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2973   }
2974 }

```

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

```

2975 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2976 \bool_if:NT \l_@@_caption_above_bool
2977 {
2978   \tl_if_empty:NF \l_@@_caption_tl
2979   {
2980     \bool_set_false:N \g_@@_caption_finished_bool
2981     \int_gzero:N \c@tabularnote
2982     \@@_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).

```

2983 \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
2984 {
2985   \tl_gput_right:Nx \g_@@_aux_tl
2986   {
2987     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2988     { \int_use:N \g_@@_notes_caption_int }
2989   }
2990   \int_gzero:N \g_@@_notes_caption_int
2991 }
2992 }
2993 }

```

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

```

2994 \hbox
2995 {
2996   \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.

```

2997 \@@_create_extra_nodes:
2998 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2999 }

```

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

```

3000 \bool_lazy_any:nT
3001 {
3002   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3003   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3004   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3005 }
3006 \@@_insert_tabularnotes:
3007 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3008 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:

```

```

3009   \end { minipage }
3010 }

```

```

3011 \cs_new_protected:Npn \@@_insert_caption:
3012 {
3013   \tl_if_empty:NF \l_@@_caption_tl
3014   {
3015     \cs_if_exist:NTF \@capttype
3016     { \@@_insert_caption_i: }
3017     { \@@_error:n { caption~outside~float } }
3018   }
3019 }

```

```

3020 \cs_new_protected:Npn \@@_insert_caption_i:
3021 {
3022   \group_begin:

```

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

```

3023   \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.

```

3024   \IfPackageLoadedTF { floatrow }
3025   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3026   { }
3027   \tl_if_empty:NTF \l_@@_short_caption_tl
3028   { \caption }
3029   { \caption [ \l_@@_short_caption_tl ] }
3030   { \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.

```

3031   \bool_if:NF \g_@@_caption_finished_bool
3032   {
3033     \bool_gset_true:N \g_@@_caption_finished_bool
3034     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3035     \int_gzero:N \c@tabularnote
3036   }
3037   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3038   \group_end:
3039 }

```

```

3040 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3041 {
3042   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3043   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3044 }

```

```

3045 \cs_new_protected:Npn \@@_insert_tabularnotes:
3046 {
3047   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3048   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3049   \skip_vertical:N 0.65ex

```

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

```

3050   \group_begin:
3051   \l_@@_notes_code_before_tl
3052   \tl_if_empty:NF \g_@@_tabularnote_tl
3053   {

```

```

3054 \g_@@_tabularnote_tl \par
3055 \tl_gclear:N \g_@@_tabularnote_tl
3056 }

```

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.

```

3057 \int_compare:nNnT \c@tabularnote > \c_zero_int
3058 {
3059   \bool_if:NTF \l_@@_notes_para_bool
3060   {
3061     \begin { tabularnotes* }
3062       \seq_map_inline:Nn \g_@@_notes_seq
3063       { \@@_one_tabularnote:nn ##1 }
3064       \strut
3065     \end { tabularnotes* }

```

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

```

3066 \par
3067 }
3068 {
3069   \tabularnotes
3070   \seq_map_inline:Nn \g_@@_notes_seq
3071   { \@@_one_tabularnote:nn ##1 }
3072   \strut
3073   \endtabularnotes
3074 }
3075 }
3076 \unskip
3077 \group_end:
3078 \bool_if:NT \l_@@_notes_bottomrule_bool
3079 {
3080   \IfPackageLoadedTF { booktabs }
3081   {

```

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

```

3082 \skip_vertical:N \aboverulesep

```

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

```

3083 { \CT@arc@ \hrule height \heavyrulewidth }
3084 }
3085 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3086 }
3087 \l_@@_notes_code_after_tl
3088 \seq_gclear:N \g_@@_notes_seq
3089 \seq_gclear:N \g_@@_notes_in_caption_seq
3090 \int_gzero:N \c@tabularnote
3091 }

```

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

```

3092 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3093 {
3094   \tl_if_novalue:nTF { #1 }
3095   { \item }
3096   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3097 }

```

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`.

```

3098 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:

```

```

3099 {
3100   \pgfpicture
3101     \@@_qpoint:n { row - 1 }
3102     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3103     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3104     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3105   \endpgfpicture
3106   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3107   \int_if_zero:nT \l_@@_first_row_int
3108   {
3109     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3110     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3111   }
3112   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3113 }

```

Now, the general case.

```

3114 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3115 {

```

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

```

3116   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3117   { \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`.

```

3118   \pgfpicture
3119   \@@_qpoint:n { row - 1 }
3120   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3121   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3122   {
3123     \int_set:Nn \l_tmpa_int
3124     {
3125       \str_range:Nnn
3126         \l_@@_baseline_tl
3127         6
3128         { \tl_count:o \l_@@_baseline_tl }
3129     }
3130     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3131   }
3132   {
3133     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3134     \bool_lazy_or:nnT
3135       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3136       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3137     {
3138       \@@_error:n { bad~value~for~baseline }
3139       \int_set:Nn \l_tmpa_int 1
3140     }
3141     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3142   }
3143   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3144   \endpgfpicture
3145   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3146   \int_if_zero:nT \l_@@_first_row_int
3147   {
3148     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3149     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3150   }
3151   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3152 }

```

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.

```

3153 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3154 {

```

We will compute the real width of both delimiters used.

```

3155 \dim_zero_new:N \l_@@_real_left_delim_dim
3156 \dim_zero_new:N \l_@@_real_right_delim_dim
3157 \hbox_set:Nn \l_tmpb_box
3158 {
3159   \c_math_toggle_token
3160   \left #1
3161   \vcenter
3162   {
3163     \vbox_to_ht:nn
3164     { \box_ht_plus_dp:N \l_tmpa_box }
3165     { }
3166   }
3167   \right .
3168   \c_math_toggle_token
3169 }
3170 \dim_set:Nn \l_@@_real_left_delim_dim
3171 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3172 \hbox_set:Nn \l_tmpb_box
3173 {
3174   \c_math_toggle_token
3175   \left .
3176   \vbox_to_ht:nn
3177   { \box_ht_plus_dp:N \l_tmpa_box }
3178   { }
3179   \right #2
3180   \c_math_toggle_token
3181 }
3182 \dim_set:Nn \l_@@_real_right_delim_dim
3183 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3184 \skip_horizontal:N \l_@@_left_delim_dim
3185 \skip_horizontal:N -\l_@@_real_left_delim_dim
3186 \@@_put_box_in_flow:
3187 \skip_horizontal:N \l_@@_right_delim_dim
3188 \skip_horizontal:N -\l_@@_real_right_delim_dim
3189 }

```

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).

```

3190 \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).

```

3191 {
3192   \peek_remove_spaces:n
3193   {
3194     \peek_meaning:NTF \end
3195     \@@_analyze_end:Nn
3196     {
3197       \@@_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`).

```

3198         \exp_args:No \@@_array: \g_@@_array_preamble_tl
3199     }
3200 }
3201 }
3202 {
3203     \@@_create_col_nodes:
3204     \endarray
3205 }
```

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

```

3206 \NewDocumentEnvironment { @@-light-syntax } { b }
3207 {
```

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`.

```

3208     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3209     \tl_map_inline:nn { #1 }
3210     {
3211         \str_if_eq:nnT { ##1 } { & }
3212         { \@@_fatal:n { ampersand-in-light-syntax } }
3213         \str_if_eq:nnT { ##1 } { \ }
3214         { \@@_fatal:n { double-backslash-in-light-syntax } }
3215     }
```

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`.

```

3216     \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```

3217 }
```

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.

```

3218 {
3219     \@@_create_col_nodes:
3220     \endarray
3221 }
3222 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3223 {
3224     \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).

```

3225     \seq_clear_new:N \l_@@_rows_seq
```

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

```

3226     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3227     \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```

3228     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3229     \tl_if_empty:NF \l_tmpa_tl
3230     { \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.

```

3231 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3232 { \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`).

```

3233 \tl_build_begin:N \l_@@_new_body_tl
3234 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3235 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3236 \@@_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).

```

3237 \seq_map_inline:Nn \l_@@_rows_seq
3238 {
3239   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3240   \@@_line_with_light_syntax:n { #1 }
3241 }
3242 \tl_build_end:N \l_@@_new_body_tl
3243 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3244 {
3245   \int_set:Nn \l_@@_last_col_int
3246   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3247 }

```

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.

```

3248 \@@_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`).

```

3249 \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3250 }
3251 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3252 {
3253   \seq_clear_new:N \l_@@_cells_seq
3254   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3255   \int_set:Nn \l_@@_nb_cols_int
3256   {
3257     \int_max:nn
3258     \l_@@_nb_cols_int
3259     { \seq_count:N \l_@@_cells_seq }
3260   }
3261   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3262   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3263   \seq_map_inline:Nn \l_@@_cells_seq
3264   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3265 }
3266 \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`.

```

3267 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3268 {
3269   \str_if_eq:onT \g_@@_name_env_str { #2 }
3270   { \@@_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.

```
3271   \end { #2 }
3272 }
```

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).

```
3273 \cs_new:Npn \@@_create_col_nodes:
3274 {
3275   \crr
3276   \int_if_zero:nT \l_@@_first_col_int
3277   {
3278     \omit
3279     \hbox_overlap_left:n
3280     {
3281       \bool_if:NT \l_@@_code_before_bool
3282       { \pgfsys@markposition { \@@_env: - col - 0 } }
3283       \pgfpicture
3284       \pgfrememberpicturepositiononpagetrue
3285       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointright
3286       \str_if_empty:NF \l_@@_name_str
3287       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3288       \endpgfpicture
3289       \skip_horizontal:N 2\col@sep
3290       \skip_horizontal:N \g_@@_width_first_col_dim
3291     }
3292     &
3293   }
3294   \omit
```

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

```
3295   \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`).

```
3296   \int_if_zero:nTF \l_@@_first_col_int
3297   {
3298     \bool_if:NT \l_@@_code_before_bool
3299     {
3300       \hbox
3301       {
3302         \skip_horizontal:N -0.5\arrayrulewidth
3303         \pgfsys@markposition { \@@_env: - col - 1 }
3304         \skip_horizontal:N 0.5\arrayrulewidth
3305       }
3306     }
3307     \pgfpicture
3308     \pgfrememberpicturepositiononpagetrue
3309     \pgfcoordinate { \@@_env: - col - 1 }
3310     { \pgfpointright { - 0.5 \arrayrulewidth } \c_zero_dim }
3311     \str_if_empty:NF \l_@@_name_str
3312     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3313     \endpgfpicture
3314   }
3315   {
3316     \bool_if:NT \l_@@_code_before_bool
3317     {
3318       \hbox
3319       {
3320         \skip_horizontal:N 0.5\arrayrulewidth
3321         \pgfsys@markposition { \@@_env: - col - 1 }
3322         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3323     }
3324   }
3325   \pgfpicture
3326   \pgfrememberpicturepositiononpagetrue
3327   \pgfcoordinate { \@@_env: - col - 1 }
3328   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3329   \str_if_empty:NF \l_@@_name_str
3330   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3331   \endpgfpicture
3332 }

```

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.

```

3333   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3334   \bool_if:NF \l_@@_auto_columns_width_bool
3335   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3336   {
3337     \bool_lazy_and:nnTF
3338     \l_@@_auto_columns_width_bool
3339     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3340     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3341     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3342     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3343   }
3344   \skip_horizontal:N \g_tmpa_skip
3345   \hbox
3346   {
3347     \bool_if:NT \l_@@_code_before_bool
3348     {
3349       \hbox
3350       {
3351         \skip_horizontal:N -0.5\arrayrulewidth
3352         \pgfsys@markposition { \@@_env: - col - 2 }
3353         \skip_horizontal:N 0.5\arrayrulewidth
3354       }
3355     }
3356     \pgfpicture
3357     \pgfrememberpicturepositiononpagetrue
3358     \pgfcoordinate { \@@_env: - col - 2 }
3359     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3360     \str_if_empty:NF \l_@@_name_str
3361     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3362     \endpgfpicture
3363   }

```

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.

```

3364   \int_gset_eq:NN \g_tmpa_int \c_one_int
3365   \bool_if:NTF \g_@@_last_col_found_bool
3366   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3367   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3368   {
3369     &
3370     \omit
3371     \int_gincr:N \g_tmpa_int

```

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

```

3372     \skip_horizontal:N \g_tmpa_skip
3373     \bool_if:NT \l_@@_code_before_bool
3374     {

```

```

3375         \hbox
3376         {
3377             \skip_horizontal:N -0.5\arrayrulewidth
3378             \pgfsys@markposition
3379             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3380             \skip_horizontal:N 0.5\arrayrulewidth
3381         }
3382     }

```

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

```

3383     \pgfpicture
3384     \pgfrememberpicturepositiononpagetrue
3385     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3386     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3387     \str_if_empty:NF \l_@@_name_str
3388     {
3389         \pgfnodealias
3390         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3391         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3392     }
3393     \endpgfpicture
3394 }

```

```

3395     &
3396     \omit

```

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

```

3397     \int_if_zero:nT \g_@@_col_total_int
3398     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3399     \skip_horizontal:N \g_tmpa_skip
3400     \int_gincr:N \g_tmpa_int
3401     \bool_lazy_any:nF % modified 2023/12/13
3402     {
3403         \g_@@_delims_bool
3404         \l_@@_tabular_bool
3405         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3406         \l_@@_exterior_arraycolsep_bool
3407         \l_@@_bar_at_end_of_pream_bool
3408     }
3409     { \skip_horizontal:N -\col@sep }
3410     \bool_if:NT \l_@@_code_before_bool
3411     {
3412         \hbox
3413         {
3414             \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.

```

3415         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3416         { \skip_horizontal:N -\arraycolsep }
3417         \pgfsys@markposition
3418         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3419         \skip_horizontal:N 0.5\arrayrulewidth
3420         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3421         { \skip_horizontal:N \arraycolsep }
3422     }
3423 }
3424 \pgfpicture
3425 \pgfrememberpicturepositiononpagetrue
3426 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3427 {
3428     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3429         {
3430             \pgfpoint
3431             { - 0.5 \arrayrulewidth - \arraycolsep }
3432             \c_zero_dim
3433         }
3434         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3435     }
3436     \str_if_empty:NF \l_@@_name_str
3437     {
3438         \pgfnodealias
3439         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3440         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3441     }
3442     \endpgfpicture

3443     \bool_if:NT \g_@@_last_col_found_bool
3444     {
3445         \hbox_overlap_right:n
3446         {
3447             \skip_horizontal:N \g_@@_width_last_col_dim
3448             \skip_horizontal:N \col@sep % added 2023-11-05
3449             \bool_if:NT \l_@@_code_before_bool
3450             {
3451                 \pgfsys@markposition
3452                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3453             }
3454             \pgfpicture
3455             \pgfrememberpicturepositiononpagetrue
3456             \pgfcoordinate
3457             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3458             \pgfpointorigin
3459             \str_if_empty:NF \l_@@_name_str
3460             {
3461                 \pgfnodealias
3462                 {
3463                     \l_@@_name_str - col
3464                     - \int_eval:n { \g_@@_col_total_int + 1 }
3465                 }
3466                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3467             }
3468             \endpgfpicture
3469         }
3470     }
3471     \cr
3472 }

```

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

```

3473 \tl_const:Nn \c_@@_preamble_first_col_tl
3474 {
3475     >
3476     {

```

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).

```

3477     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3478     \bool_gset_true:N \g_@@_after_col_zero_bool
3479     \@@_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.

```

3480     \hbox_set:Nw \l_@@_cell_box
3481     \@@_math_toggle:
3482     \@@_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”.

```

3483     \int_compare:nNnT \c@iRow > \c_zero_int
3484     {
3485         \bool_lazy_or:nnT
3486         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3487         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3488         {
3489             \l_@@_code_for_first_col_tl
3490             \xglobal \colorlet { nicematrix-first-col } { . }
3491         }
3492     }
3493 }

```

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`.

```

3494     l
3495     <
3496     {
3497         \@@_math_toggle:
3498         \hbox_set_end:
3499         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3500         \@@_adjust_size_box:
3501         \@@_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.

```

3502     \dim_gset:Nn \g_@@_width_first_col_dim
3503     { \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.

```

3504     \hbox_overlap_left:n
3505     {
3506         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3507         \@@_node_for_cell:
3508         { \box_use_drop:N \l_@@_cell_box }
3509         \skip_horizontal:N \l_@@_left_delim_dim
3510         \skip_horizontal:N \l_@@_left_margin_dim
3511         \skip_horizontal:N \l_@@_extra_left_margin_dim
3512     }
3513     \bool_gset_false:N \g_@@_empty_cell_bool
3514     \skip_horizontal:N -2\col@sep
3515 }
3516 }

```

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

```

3517 \tl_const:Nn \c_@@_preamble_last_col_tl
3518 {
3519     >
3520     {
3521         \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).

```

3522     \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.

```

3523     \bool_gset_true:N \g_@@_last_col_found_bool
3524     \int_gincr:N \c@jCol
3525     \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.

```

3526     \hbox_set:Nw \l_@@_cell_box
3527     \@@_math_toggle:
3528     \@@_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”.

```

3529     \int_compare:nNnT \c@iRow > \c_zero_int
3530     {
3531         \bool_lazy_or:nnT
3532         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3533         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3534         {
3535             \l_@@_code_for_last_col_tl
3536             \xglobal \colorlet { nicematrix-last-col } { . }
3537         }
3538     }
3539 }
3540 1
3541 <
3542 {
3543     \@@_math_toggle:
3544     \hbox_set_end:
3545     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3546     \@@_adjust_size_box:
3547     \@@_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.

```

3548     \dim_gset:Nn \g_@@_width_last_col_dim
3549     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3550     \skip_horizontal:N -2\col@sep

```

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

```

3551     \hbox_overlap_right:n
3552     {
3553         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3554         {
3555             \skip_horizontal:N \l_@@_right_delim_dim
3556             \skip_horizontal:N \l_@@_right_margin_dim
3557             \skip_horizontal:N \l_@@_extra_right_margin_dim
3558             \@@_node_for_cell:
3559         }
3560     }
3561     \bool_gset_false:N \g_@@_empty_cell_bool
3562 }
3563 }

```

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

```

3564 \NewDocumentEnvironment { NiceArray } { }
3565 {
3566     \bool_gset_false:N \g_@@_delims_bool
3567     \str_if_empty:NT \g_@@_name_env_str
3568     { \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).

```

3569     \NiceArrayWithDelims . .
3570 }
3571 { \endNiceArrayWithDelims }

```

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

```

3572 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3573 {
3574     \NewDocumentEnvironment { #1 NiceArray } { }
3575     {

```

```

3576 \bool_gset_true:N \g_@@_delims_bool
3577 \str_if_empty:NT \g_@@_name_env_str
3578 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3579 \@@_test_if_math_mode:
3580 \NiceArrayWithDelims #2 #3
3581 }
3582 { \endNiceArrayWithDelims }
3583 }
3584 \@@_def_env:nnn p ( )
3585 \@@_def_env:nnn b [ ]
3586 \@@_def_env:nnn B \{ \}
3587 \@@_def_env:nnn v | |
3588 \@@_def_env:nnn V \l \l

```

## 14 The environment `{NiceMatrix}` and its variants

```

3589 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3590 {
3591   \bool_set_false:N \l_@@_preamble_bool
3592   \tl_clear:N \l_tmpa_tl
3593   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3594   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3595   \tl_put_right:Nn \l_tmpa_tl
3596   {
3597     *
3598     {
3599       \int_case:nnF \l_@@_last_col_int
3600       {
3601         { -2 } { \c@MaxMatrixCols }
3602         { -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).

```

3603       }
3604       { \int_eval:n { \l_@@_last_col_int - 1 } }
3605     }
3606     { #2 }
3607   }
3608   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3609   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3610 }
3611 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3612 \clist_map_inline:nn { p , b , B , v , V }
3613 {
3614   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3615   {
3616     \bool_gset_true:N \g_@@_delims_bool
3617     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3618     % added 2023/10/01
3619     \int_if_zero:nT \l_@@_last_col_int
3620     {
3621       \bool_set_true:N \l_@@_last_col_without_value_bool
3622       \int_set:Nn \l_@@_last_col_int { -1 }
3623     }
3624     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3625     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3626   }
3627   { \use:c { end #1 NiceArray } }
3628 }

```

We define also an environment `{NiceMatrix}`

```

3629 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3630 {
3631   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3632   % added 2023/10/01
3633   \int_if_zero:nT \l_@@_last_col_int
3634   {
3635     \bool_set_true:N \l_@@_last_col_without_value_bool
3636     \int_set:Nn \l_@@_last_col_int { -1 }
3637   }
3638   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3639   \bool_lazy_or:nnT
3640     { \clist_if_empty_p:N \l_@@_vlines_clist }
3641     { \l_@@_except_borders_bool }
3642     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3643   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3644 }
3645 { \endNiceArray }

```

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

```

3646 \cs_new_protected:Npn \@@_NotEmpty:
3647 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3648 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! O { } }
3649 {

```

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`.

```

3650   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3651     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3652   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3653   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3654   \tl_if_empty:NF \l_@@_short_caption_tl
3655   {
3656     \tl_if_empty:NT \l_@@_caption_tl
3657     {
3658       \@@_error_or_warning:n { short-caption~without~caption }
3659       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3660     }
3661   }
3662   \tl_if_empty:NF \l_@@_label_tl
3663   {
3664     \tl_if_empty:NT \l_@@_caption_tl
3665     { \@@_error_or_warning:n { label~without~caption } }
3666   }
3667   \NewDocumentEnvironment { TabularNote } { b }
3668   {
3669     \bool_if:NTF \l_@@_in_code_after_bool
3670       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3671       {
3672         \tl_if_empty:NF \g_@@_tabularnote_tl
3673         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3674         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3675       }
3676   }
3677   { }
3678   \@@_settings_for_tabular:
3679   \NiceArray { #2 }
3680 }
3681 { \endNiceArray }

```

```

3682 \cs_new_protected:Npn \@@_settings_for_tabular:
3683 {
3684   \bool_set_true:N \l_@@_tabular_bool
3685   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3686   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3687   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3688 }

3689 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3690 {
3691   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3692   \dim_zero_new:N \l_@@_width_dim
3693   \dim_set:Nn \l_@@_width_dim { #1 }
3694   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3695   \@@_settings_for_tabular:
3696   \NiceArray { #3 }
3697 }
3698 {
3699   \endNiceArray
3700   \int_if_zero:nT \g_@@_total_X_weight_int
3701     { \@@_error:n { NiceTabularX~without~X } }
3702 }

3703 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3704 {
3705   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3706   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3707   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3708   \@@_settings_for_tabular:
3709   \NiceArray { #3 }
3710 }
3711 { \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`).

```

3712 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3713 {
3714   \bool_lazy_all:nT
3715     {
3716       { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3717       \l_@@_hvlines_bool
3718       { ! \g_@@_delims_bool }
3719       { ! \l_@@_except_borders_bool }
3720     }
3721     {
3722       \bool_set_true:N \l_@@_except_borders_bool
3723       \clist_if_empty:NF \l_@@_corners_clist
3724         { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3725       \tl_gput_right:Nn \g_@@_pre_code_after_tl
3726         {
3727           \@@_stroke_block:nnn
3728             {
3729               rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3730               draw = \l_@@_rules_color_tl
3731             }
3732           { 1-1 }

```

```

3733         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3734     }
3735 }
3736 }

```

```

3737 \cs_new_protected:Npn \@@_after_array:
3738 {
3739     \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.

```

3740     \bool_if:NT \g_@@_last_col_found_bool
3741     { \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`.

```

3742     \bool_if:NT \l_@@_last_col_without_value_bool
3743     { \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.

```

3744     \bool_if:NT \l_@@_last_row_without_value_bool
3745     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3746 \tl_gput_right:Nx \g_@@_aux_tl
3747 {
3748     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3749     {
3750         \int_use:N \l_@@_first_row_int ,
3751         \int_use:N \c@iRow ,
3752         \int_use:N \g_@@_row_total_int ,
3753         \int_use:N \l_@@_first_col_int ,
3754         \int_use:N \c@jCol ,
3755         \int_use:N \g_@@_col_total_int
3756     }
3757 }

```

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`).

```

3758     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3759     {
3760         \tl_gput_right:Nx \g_@@_aux_tl
3761         {
3762             \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3763             { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3764         }
3765     }
3766 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3767 {
3768     \tl_gput_right:Nx \g_@@_aux_tl
3769     {
3770         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3771         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3772         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3773         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3774     }
3775 }

```

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

```

3776     \@@_create_diag_nodes:

```

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

```

3777 \pgfpicture
3778 \int_step_inline:nn \c@iRow
3779 {
3780   \pgfnodealias
3781   { \@@_env: - ##1 - last }
3782   { \@@_env: - ##1 - \int_use:N \c@jCol }
3783 }
3784 \int_step_inline:nn \c@jCol
3785 {
3786   \pgfnodealias
3787   { \@@_env: - last - ##1 }
3788   { \@@_env: - \int_use:N \c@iRow - ##1 }
3789 }
3790 \str_if_empty:NF \l_@@_name_str
3791 {
3792   \int_step_inline:nn \c@iRow
3793   {
3794     \pgfnodealias
3795     { \l_@@_name_str - ##1 - last }
3796     { \@@_env: - ##1 - \int_use:N \c@jCol }
3797   }
3798   \int_step_inline:nn \c@jCol
3799   {
3800     \pgfnodealias
3801     { \l_@@_name_str - last - ##1 }
3802     { \@@_env: - \int_use:N \c@iRow - ##1 }
3803   }
3804 }
3805 \endpgfpicture

```

By default, the diagonal lines will be parallelized<sup>11</sup>. 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.

```

3806 \bool_if:NT \l_@@_parallelize_diags_bool
3807 {
3808   \int_gzero_new:N \g_@@_ddots_int
3809   \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the  $\Delta_x$  and  $\Delta_y$  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  $\Delta_x$  and  $\Delta_y$  of the first `\Iddots` diagonal.

```

3810 \dim_gzero_new:N \g_@@_delta_x_one_dim
3811 \dim_gzero_new:N \g_@@_delta_y_one_dim
3812 \dim_gzero_new:N \g_@@_delta_x_two_dim
3813 \dim_gzero_new:N \g_@@_delta_y_two_dim
3814 }
3815 \int_zero_new:N \l_@@_initial_i_int
3816 \int_zero_new:N \l_@@_initial_j_int
3817 \int_zero_new:N \l_@@_final_i_int
3818 \int_zero_new:N \l_@@_final_j_int
3819 \bool_set_false:N \l_@@_initial_open_bool
3820 \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.

```

3821 \bool_if:NT \l_@@_small_bool
3822 {

```

---

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

```

3823 \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3824 \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.

```

3825 \dim_set:Nn \l_@@_xdots_shorten_start_dim
3826 { 0.6 \l_@@_xdots_shorten_start_dim }
3827 \dim_set:Nn \l_@@_xdots_shorten_end_dim
3828 { 0.6 \l_@@_xdots_shorten_end_dim }
3829 }

```

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

```

3830 \@@_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.

```

3831 \@@_compute_corners:

```

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

```

3832 \@@_adjust_pos_of_blocks_seq:
3833 \@@_deal_with_rounded_corners:
3834 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3835 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3836 \IfPackageLoadedTF { tikz }
3837 {
3838   \tikzset
3839   {
3840     every-picture / .style =
3841     {
3842       overlay ,
3843       remember~picture ,
3844       name~prefix = \@@_env: -
3845     }
3846   }
3847 }
3848 { }
3849 \cs_set_eq:NN \ialign \@@_old_ialign:
3850 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3851 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3852 \cs_set_eq:NN \OverBrace \@@_OverBrace
3853 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3854 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3855 \cs_set_eq:NN \line \@@_line
3856 \g_@@_pre_code_after_tl
3857 \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.

```

3858 \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).

```

3859 \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`).

```
3860 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3861 { \@@_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:`.

```
3862 \bool_set_true:N \l_@@_in_code_after_bool
3863 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3864 \scan_stop:
3865 \tl_gclear:N \g_nicematrix_code_after_tl
3866 \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.

```
3867 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3868 \tl_if_empty:NF \g_@@_pre_code_before_tl
3869 {
3870   \tl_gput_right:Nx \g_@@_aux_tl
3871   {
3872     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3873     { \exp_not:o \g_@@_pre_code_before_tl }
3874   }
3875   \tl_gclear:N \g_@@_pre_code_before_tl
3876 }
3877 \tl_if_empty:NF \g_nicematrix_code_before_tl
3878 {
3879   \tl_gput_right:Nx \g_@@_aux_tl
3880   {
3881     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3882     { \exp_not:o \g_nicematrix_code_before_tl }
3883   }
3884   \tl_gclear:N \g_nicematrix_code_before_tl
3885 }

3886 \str_gclear:N \g_@@_name_env_str
3887 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array<sup>12</sup>. 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`.

```
3888 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3889 }
```

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`.

```
3890 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3891 { \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

---

<sup>12</sup>e.g. `\color[rgb]{0.5,0.5,0}`

`\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).

```

3892 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3893 {
3894   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3895   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3896 }

```

The following command must *not* be protected.

```

3897 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3898 {
3899   { #1 }
3900   { #2 }
3901   {
3902     \int_compare:nNnTF { #3 } > { 99 }
3903     { \int_use:N \c@iRow }
3904     { #3 }
3905   }
3906   {
3907     \int_compare:nNnTF { #4 } > { 99 }
3908     { \int_use:N \c@jCol }
3909     { #4 }
3910   }
3911   { #5 }
3912 }

```

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).

```

3913 \hook_gput_code:nnn { begindocument } { . }
3914 {
3915   \cs_new_protected:Npx \@@_draw_dotted_lines:
3916   {
3917     \c_@@_pgfortikzpicture_tl
3918     \@@_draw_dotted_lines_i:
3919     \c_@@_endpgfortikzpicture_tl
3920   }
3921 }

```

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

```

3922 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3923 {
3924   \pgfrememberpicturepositiononpagetrue
3925   \pgf@relevantforpicturesizefalse
3926   \g_@@_HVdotsfor_lines_tl
3927   \g_@@_Vdots_lines_tl
3928   \g_@@_Ddots_lines_tl
3929   \g_@@_Iddots_lines_tl
3930   \g_@@_Cdots_lines_tl
3931   \g_@@_Ldots_lines_tl
3932 }

3933 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3934 {
3935   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3936   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3937 }

```

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

```

3938 \pgfdeclareshape { @@_diag_node }
3939 {
3940   \savedanchor { \five }
3941   {
3942     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3943     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3944   }
3945   \anchor { 5 } { \five }
3946   \anchor { center } { \pgfpointorigin }
3947 }

```

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).

```

3948 \cs_new_protected:Npn \@@_create_diag_nodes:
3949 {
3950   \pgfpicture
3951   \pgfrememberpicturepositiononpagetrue
3952   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3953   {
3954     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3955     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3956     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3957     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3958     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3959     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3960     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3961     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3962     \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.

```

3963     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3964     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3965     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3966     \str_if_empty:NF \l_@@_name_str
3967     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3968   }

```

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

```

3969   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3970   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3971   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3972   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3973   \pgfcoordinate
3974   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3975   \pgfnodealias
3976   { \@@_env: - last }
3977   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3978   \str_if_empty:NF \l_@@_name_str
3979   {
3980     \pgfnodealias
3981     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3982     { \@@_env: - \int_use:N \l_tmpa_int }
3983     \pgfnodealias
3984     { \l_@@_name_str - last }
3985     { \@@_env: - last }
3986   }
3987 \endpgfpicture
3988 }

```

## 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.

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

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.

```
3989 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3990 {
```

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

```
3991 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3992 \int_set:Nn \l_@@_initial_i_int { #1 }
3993 \int_set:Nn \l_@@_initial_j_int { #2 }
3994 \int_set:Nn \l_@@_final_i_int { #1 }
3995 \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.

```
3996 \bool_set_false:N \l_@@_stop_loop_bool
3997 \bool_do_until:Nn \l_@@_stop_loop_bool
3998 {
3999   \int_add:Nn \l_@@_final_i_int { #3 }
4000   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4001 \bool_set_false:N \l_@@_final_open_bool
4002 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4003 {
4004   \int_compare:nNnTF { #3 } = \c_one_int
4005   { \bool_set_true:N \l_@@_final_open_bool }
4006   {
4007     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
4008     { \bool_set_true:N \l_@@_final_open_bool }
4009   }
4010 }
4011 {
4012   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
```

```

4013     {
4014         \int_compare:nNnT { #4 } = { -1 }
4015         { \bool_set_true:N \l_@@_final_open_bool }
4016     }
4017     {
4018         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4019         {
4020             \int_compare:nNnT { #4 } = \c_one_int
4021             { \bool_set_true:N \l_@@_final_open_bool }
4022         }
4023     }
4024 }
4025 \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.

```

4026     {

```

We do a step backwards.

```

4027         \int_sub:Nn \l_@@_final_i_int { #3 }
4028         \int_sub:Nn \l_@@_final_j_int { #4 }
4029         \bool_set_true:N \l_@@_stop_loop_bool
4030     }

```

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`.

```

4031     {
4032         \cs_if_exist:cTF
4033         {
4034             @@ _ dotted _
4035             \int_use:N \l_@@_final_i_int -
4036             \int_use:N \l_@@_final_j_int
4037         }
4038         {
4039             \int_sub:Nn \l_@@_final_i_int { #3 }
4040             \int_sub:Nn \l_@@_final_j_int { #4 }
4041             \bool_set_true:N \l_@@_final_open_bool
4042             \bool_set_true:N \l_@@_stop_loop_bool
4043         }
4044         {
4045             \cs_if_exist:cTF
4046             {
4047                 pgf @ sh @ ns @ \@@_env:
4048                 - \int_use:N \l_@@_final_i_int
4049                 - \int_use:N \l_@@_final_j_int
4050             }
4051             { \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.

```

4052         {
4053             \cs_set:cpn
4054             {
4055                 @@ _ dotted _
4056                 \int_use:N \l_@@_final_i_int -
4057                 \int_use:N \l_@@_final_j_int
4058             }
4059             { }
4060         }
4061     }
4062 }
4063 }

```

For \l\_@@\_initial\_i\_int and \l\_@@\_initial\_j\_int the programming is similar to the previous one.

```

4064 \bool_set_false:N \l_@@_stop_loop_bool
4065 \bool_do_until:Nn \l_@@_stop_loop_bool
4066 {
4067   \int_sub:Nn \l_@@_initial_i_int { #3 }
4068   \int_sub:Nn \l_@@_initial_j_int { #4 }
4069   \bool_set_false:N \l_@@_initial_open_bool
4070   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4071   {
4072     \int_compare:nNnTF { #3 } = \c_one_int
4073     { \bool_set_true:N \l_@@_initial_open_bool }
4074     {
4075       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4076       { \bool_set_true:N \l_@@_initial_open_bool }
4077     }
4078   }
4079   {
4080     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4081     {
4082       \int_compare:nNnT { #4 } = \c_one_int
4083       { \bool_set_true:N \l_@@_initial_open_bool }
4084     }
4085     {
4086       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4087       {
4088         \int_compare:nNnT { #4 } = { -1 }
4089         { \bool_set_true:N \l_@@_initial_open_bool }
4090       }
4091     }
4092   }
4093   \bool_if:NTF \l_@@_initial_open_bool
4094   {
4095     \int_add:Nn \l_@@_initial_i_int { #3 }
4096     \int_add:Nn \l_@@_initial_j_int { #4 }
4097     \bool_set_true:N \l_@@_stop_loop_bool
4098   }
4099   {
4100     \cs_if_exist:cTF
4101     {
4102       @@ _ dotted _
4103       \int_use:N \l_@@_initial_i_int -
4104       \int_use:N \l_@@_initial_j_int
4105     }
4106     {
4107       \int_add:Nn \l_@@_initial_i_int { #3 }
4108       \int_add:Nn \l_@@_initial_j_int { #4 }
4109       \bool_set_true:N \l_@@_initial_open_bool
4110       \bool_set_true:N \l_@@_stop_loop_bool
4111     }
4112     {
4113       \cs_if_exist:cTF
4114       {
4115         pgf @ sh @ ns @ \@@_env:
4116         - \int_use:N \l_@@_initial_i_int
4117         - \int_use:N \l_@@_initial_j_int
4118       }
4119       { \bool_set_true:N \l_@@_stop_loop_bool }
4120       {
4121         \cs_set:cpn
4122         {
4123           @@ _ dotted _
4124           \int_use:N \l_@@_initial_i_int -

```

```

4125             \int_use:N \l_@@_initial_j_int
4126         }
4127     { }
4128 }
4129 }
4130 }
4131 }

```

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.

```

4132 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4133 {
4134     { \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`.

```

4135     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4136     { \int_use:N \l_@@_final_i_int }
4137     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4138     { } % for the name of the block
4139 }
4140 }

```

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.

```

4141 \cs_new_protected:Npn \@@_open_shorten:
4142 {
4143     \bool_if:NT \l_@@_initial_open_bool
4144     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4145     \bool_if:NT \l_@@_final_open_bool
4146     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4147 }

```

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).

```

4148 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4149 {
4150     \int_set:Nn \l_@@_row_min_int 1
4151     \int_set:Nn \l_@@_col_min_int 1
4152     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4153     \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`.

```

4154 \seq_map_inline:Nn \g_@@_submatrix_seq
4155 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4156 }

```

#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.

```

4157 \set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4158 {
4159     \int_compare:nNf { #3 } > { #1 }
4160     {
4161         \int_compare:nNf { #1 } > { #5 }
4162         {

```

```

4163         \int_compare:nNnF { #4 } > { #2 }
4164         {
4165             \int_compare:nNnF { #2 } > { #6 }
4166             {
4167                 \int_set:Nn \l_@@_row_min_int
4168                 { \int_max:nn \l_@@_row_min_int { #3 } }
4169                 \int_set:Nn \l_@@_col_min_int
4170                 { \int_max:nn \l_@@_col_min_int { #4 } }
4171                 \int_set:Nn \l_@@_row_max_int
4172                 { \int_min:nn \l_@@_row_max_int { #5 } }
4173                 \int_set:Nn \l_@@_col_max_int
4174                 { \int_min:nn \l_@@_col_max_int { #6 } }
4175             }
4176         }
4177     }
4178 }
4179 }

4180 \cs_new_protected:Npn \@@_set_initial_coords:
4181 {
4182     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4183     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4184 }
4185 \cs_new_protected:Npn \@@_set_final_coords:
4186 {
4187     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4188     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4189 }
4190 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4191 {
4192     \pgfpointanchor
4193     {
4194         \@@_env:
4195         - \int_use:N \l_@@_initial_i_int
4196         - \int_use:N \l_@@_initial_j_int
4197     }
4198     { #1 }
4199     \@@_set_initial_coords:
4200 }
4201 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4202 {
4203     \pgfpointanchor
4204     {
4205         \@@_env:
4206         - \int_use:N \l_@@_final_i_int
4207         - \int_use:N \l_@@_final_j_int
4208     }
4209     { #1 }
4210     \@@_set_final_coords:
4211 }
4212 \cs_new_protected:Npn \@@_open_x_initial_dim:
4213 {
4214     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4215     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4216     {
4217         \cs_if_exist:cT
4218         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4219         {
4220             \pgfpointanchor
4221             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4222             { west }
4223             \dim_set:Nn \l_@@_x_initial_dim
4224             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }

```

```

4225     }
4226 }

```

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

```

4227 \dim_compare:nNt \l_@@_x_initial_dim = \c_max_dim
4228 {
4229   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4230   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4231   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4232 }
4233 }
4234 \cs_new_protected:Npn \@@_open_x_final_dim:
4235 {
4236   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4237   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4238   {
4239     \cs_if_exist:cT
4240     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4241     {
4242       \pgfpoinanchor
4243       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4244       { east }
4245       \dim_set:Nn \l_@@_x_final_dim
4246       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4247     }
4248   }

```

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

```

4249 \dim_compare:nNt \l_@@_x_final_dim = { - \c_max_dim }
4250 {
4251   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4252   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4253   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4254 }
4255 }

```

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.

```

4256 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4257 {
4258   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4259   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4260   {
4261     \@@_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.

```

4262 \group_begin:
4263 \@@_open_shorten:
4264 \int_if_zero:nTF { #1 }
4265 { \color { nicematrix-first-row } }
4266 {

```

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.

```

4267 \int_compare:nNt { #1 } = \l_@@_last_row_int
4268 { \color { nicematrix-last-row } }
4269 }
4270 \keys_set:nn { NiceMatrix / xdots } { #3 }
4271 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4272 \@@_actually_draw_Ldots:
4273 \group_end:
4274 }
4275 }

```

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

The following function is also used by `\Hdotsfor`.

```

4276 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4277 {
4278   \bool_if:NTF \l_@@_initial_open_bool
4279   {
4280     \@@_open_x_initial_dim:
4281     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4282     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4283   }
4284   { \@@_set_initial_coords_from_anchor:n { base~east } }
4285   \bool_if:NTF \l_@@_final_open_bool
4286   {
4287     \@@_open_x_final_dim:
4288     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4289     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4290   }
4291   { \@@_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.

```

4292   \bool_lazy_all:nTF
4293   {
4294     \l_@@_initial_open_bool
4295     \l_@@_final_open_bool
4296     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4297   }
4298   {
4299     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4300     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4301   }

```

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 (?).

```

4302   {
4303     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4304     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4305   }
4306   \@@_draw_line:
4307 }

```

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.

```

4308 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4309 {
4310   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4311   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4312   {
4313     \@@_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.

```

4314     \group_begin:
4315     \@@_open_shorten:
4316     \int_if_zero:nTF { #1 }
4317     { \color { nicematrix-first-row } }
4318     {

```

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.

```

4319         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4320         { \color { nicematrix-last-row } }
4321     }
4322     \keys_set:nn { NiceMatrix / xdots } { #3 }
4323     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4324     \@@_actually_draw_Cdots:
4325     \group_end:
4326 }
4327 }

```

The command `\@@_actually_draw_Cdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

```

4328 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4329 {
4330     \bool_if:NTF \l_@@_initial_open_bool
4331     { \@@_open_x_initial_dim: }
4332     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4333     \bool_if:NTF \l_@@_final_open_bool
4334     { \@@_open_x_final_dim: }
4335     { \@@_set_final_coords_from_anchor:n { mid-west } }
4336     \bool_lazy_and:nnTF
4337     \l_@@_initial_open_bool
4338     \l_@@_final_open_bool
4339     {
4340         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4341         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4342         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4343         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4344         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4345     }
4346     {
4347         \bool_if:NT \l_@@_initial_open_bool
4348         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4349         \bool_if:NT \l_@@_final_open_bool
4350         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4351     }
4352     \@@_draw_line:
4353 }
4354 \cs_new_protected:Npn \@@_open_y_initial_dim:
4355 {
4356     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4357     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4358     {

```

```

4359 \cs_if_exist:cT
4360 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4361 {
4362     \pgfpointanchor
4363     { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4364     { north }
4365     \dim_set:Nn \l_@@_y_initial_dim
4366     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4367 }
4368 }
4369 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4370 {
4371     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4372     \dim_set:Nn \l_@@_y_initial_dim
4373     {
4374         \fp_to_dim:n
4375         {
4376             \pgf@y
4377             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4378         }
4379     }
4380 }
4381 }
4382 \cs_new_protected:Npn \@@_open_y_final_dim:
4383 {
4384     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4385     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4386     {
4387         \cs_if_exist:cT
4388         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4389         {
4390             \pgfpointanchor
4391             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4392             { south }
4393             \dim_set:Nn \l_@@_y_final_dim
4394             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4395         }
4396     }
4397     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4398     {
4399         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4400         \dim_set:Nn \l_@@_y_final_dim
4401         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4402     }
4403 }

```

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.

```

4404 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4405 {
4406     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4407     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4408     {
4409         \@@_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.

```

4410 \group_begin:
4411 \@@_open_shorten:
4412 \int_if_zero:nTF { #2 }
4413 { \color { nicematrix-first-col } }
4414 {
4415     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4416     { \color { nicematrix-last-col } }

```

```

4417     }
4418     \keys_set:nn { NiceMatrix / xdots } { #3 }
4419     \tl_if_empty:oF \l_@@_xdots_color_tl
4420     { \color { \l_@@_xdots_color_tl } }
4421     \@@_actually_draw_Vdots:
4422     \group_end:
4423   }
4424 }

```

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`.

```

4425 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4426 {

```

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

```

4427     \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.

```

4428     {
4429       \@@_open_y_initial_dim:
4430       \@@_open_y_final_dim:
4431       \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4432     {
4433       \@@_qpoint:n { col - 1 }
4434       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4435       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4436       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4437       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4438     }
4439     {
4440       \bool_lazy_and:nnTF
4441       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4442       { \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”.

```

4443     {
4444       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4445       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4446       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4447       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4448       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4449     }

```

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

```

4450     {
4451       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4452       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4453       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4454       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4455     }
4456   }
4457 }

```

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 `l` or may be considered as if.

```

4458 {
4459     \bool_set_false:N \l_tmpa_bool
4460     \bool_if:NF \l_@@_initial_open_bool
4461     {
4462         \bool_if:NF \l_@@_final_open_bool
4463         {
4464             \@@_set_initial_coords_from_anchor:n { south-west }
4465             \@@_set_final_coords_from_anchor:n { north-west }
4466             \bool_set:Nn \l_tmpa_bool
4467                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4468         }
4469     }

```

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

```

4470     \bool_if:NTF \l_@@_initial_open_bool
4471     {
4472         \@@_open_y_initial_dim:
4473         \@@_set_final_coords_from_anchor:n { north }
4474         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4475     }
4476     {
4477         \@@_set_initial_coords_from_anchor:n { south }
4478         \bool_if:NTF \l_@@_final_open_bool
4479         \@@_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.

```

4480     {
4481         \@@_set_final_coords_from_anchor:n { north }
4482         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4483         {
4484             \dim_set:Nn \l_@@_x_initial_dim
4485             {
4486                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4487                     \l_@@_x_initial_dim \l_@@_x_final_dim
4488             }
4489         }
4490     }
4491 }
4492 }
4493 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4494 \@@_draw_line:
4495 }

```

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.

```

4496 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4497 {
4498     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4499     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4500     {
4501         \@@_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.

```

4502     \group_begin:
4503     \@@_open_shorten:

```

```

4504         \keys_set:nn { NiceMatrix / xdots } { #3 }
4505         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4506         \@@_actually_draw_Ddots:
4507     \group_end:
4508 }
4509 }

```

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.`

```

4510 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4511 {
4512     \bool_if:NTF \l_@@_initial_open_bool
4513     {
4514         \@@_open_y_initial_dim:
4515         \@@_open_x_initial_dim:
4516     }
4517     { \@@_set_initial_coords_from_anchor:n { south~east } }
4518     \bool_if:NTF \l_@@_final_open_bool
4519     {
4520         \@@_open_x_final_dim:
4521         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4522     }
4523     { \@@_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.

```

4524     \bool_if:NT \l_@@_parallelize_diags_bool
4525     {
4526         \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).

```

4527     \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  $\Delta_x$  and the  $\Delta_y$  of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

4528     {
4529         \dim_gset:Nn \g_@@_delta_x_one_dim
4530         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4531         \dim_gset:Nn \g_@@_delta_y_one_dim
4532         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4533     }

```

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`.

```

4534     {
4535         \dim_set:Nn \l_@@_y_final_dim
4536         {
4537             \l_@@_y_initial_dim +
4538             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4539             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4540         }
4541     }
4542 }
4543 \@@_draw_line:
4544 }

```

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.

```

4545 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4546 {
4547   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4548   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4549   {
4550     \@@_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.

```

4551   \group_begin:
4552   \@@_open_shorten:
4553   \keys_set:nn { NiceMatrix / xdots } { #3 }
4554   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4555   \@@_actually_draw_Iddots:
4556   \group_end:
4557 }
4558 }

```

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`.

```

4559 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4560 {
4561   \bool_if:NTF \l_@@_initial_open_bool
4562   {
4563     \@@_open_y_initial_dim:
4564     \@@_open_x_initial_dim:
4565   }
4566   { \@@_set_initial_coords_from_anchor:n { south~west } }
4567   \bool_if:NTF \l_@@_final_open_bool
4568   {
4569     \@@_open_y_final_dim:
4570     \@@_open_x_final_dim:
4571   }
4572   { \@@_set_final_coords_from_anchor:n { north~east } }
4573   \bool_if:NT \l_@@_parallelize_diags_bool
4574   {
4575     \int_gincr:N \g_@@_iddots_int
4576     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4577     {
4578       \dim_gset:Nn \g_@@_delta_x_two_dim
4579       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4580       \dim_gset:Nn \g_@@_delta_y_two_dim
4581       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4582     }
4583     {
4584       \dim_set:Nn \l_@@_y_final_dim
4585       {
4586         \l_@@_y_initial_dim +
4587         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4588         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim

```

```

4589         }
4590     }
4591 }
4592 \@@_draw_line:
4593 }

```

## 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`

```

4594 \cs_new_protected:Npn \@@_draw_line:
4595 {
4596     \pgfrememberpicturepositiononpagetrue
4597     \pgf@relevantforpicturesizefalse
4598     \bool_lazy_or:nnTF
4599     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4600     \l_@@_dotted_bool
4601     \@@_draw_standard_dotted_line:
4602     \@@_draw_unstandard_dotted_line:
4603 }

```

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.

```

4604 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4605 {
4606     \begin { scope }
4607     \@@_draw_unstandard_dotted_line:o
4608     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4609 }

```

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

```

4610 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4611 {
4612     \@@_draw_unstandard_dotted_line:nooo
4613     { #1 }
4614     \l_@@_xdots_up_tl
4615     \l_@@_xdots_down_tl
4616     \l_@@_xdots_middle_tl
4617 }
4618 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4619 \hook_gput_code:nnn { begindocument } { . }
4620 {
4621   \IfPackageLoadedTF { tikz }
4622   {
4623     \tikzset
4624     {
4625       @@_node_above / .style = { sloped , above } ,
4626       @@_node_below / .style = { sloped , below } ,
4627       @@_node_middle / .style =
4628       {
4629         sloped ,
4630         inner-sep = \c_@@_innersep_middle_dim
4631       }
4632     }
4633   }
4634   { }
4635 }

4636 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4637 {

```

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  $\ell$  of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```

4638   \dim_zero_new:N \l_@@_l_dim
4639   \dim_set:Nn \l_@@_l_dim
4640   {
4641     \fp_to_dim:n
4642     {
4643       sqrt
4644       (
4645         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4646         +
4647         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4648       )
4649     }
4650   }

```

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.

```

4651   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4652   {
4653     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4654     \@@_draw_unstandard_dotted_line_i:
4655   }

```

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

```

4656   \bool_if:NT \l_@@_xdots_h_labels_bool
4657   {
4658     \tikzset
4659     {
4660       @@_node_above / .style = { auto = left } ,
4661       @@_node_below / .style = { auto = right } ,
4662       @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4663     }
4664   }

```

```

4665 \tl_if_empty:nF { #4 }
4666 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4667 \draw
4668 [ #1 ]
4669 ( \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).

```

4670 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4671 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4672 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4673 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4674 \end { scope }
4675 }

4676 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4677 {
4678   \dim_set:Nn \l_tmpa_dim
4679   {
4680     \l_@@_x_initial_dim
4681     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4682     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4683   }
4684   \dim_set:Nn \l_tmpb_dim
4685   {
4686     \l_@@_y_initial_dim
4687     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4688     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4689   }
4690   \dim_set:Nn \l_@@_tmpc_dim
4691   {
4692     \l_@@_x_final_dim
4693     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4694     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4695   }
4696   \dim_set:Nn \l_@@_tmpd_dim
4697   {
4698     \l_@@_y_final_dim
4699     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4700     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4701   }
4702   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4703   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4704   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4705   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4706 }

4707 \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).

```

4708 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4709 {
4710   \group_begin:

```

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

```

4711   \dim_zero_new:N \l_@@_l_dim
4712   \dim_set:Nn \l_@@_l_dim
4713   {
4714     \fp_to_dim:n
4715     {
4716       sqrt
4717       (

```

```

4718         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4719         +
4720         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4721     )
4722 }
4723 }

```

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.

```

4724 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4725 {
4726     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4727     \@@_draw_standard_dotted_line_i:
4728 }
4729 \group_end:
4730 \bool_lazy_all:nF
4731 {
4732     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4733     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4734     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4735 }
4736 \l_@@_labels_standard_dotted_line:
4737 }
4738 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4739 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4740 {

```

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

```

4741 \int_set:Nn \l_tmpa_int
4742 {
4743     \dim_ratio:nn
4744     {
4745         \l_@@_l_dim
4746         - \l_@@_xdots_shorten_start_dim
4747         - \l_@@_xdots_shorten_end_dim
4748     }
4749     \l_@@_xdots_inter_dim
4750 }

```

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

```

4751 \dim_set:Nn \l_tmpa_dim
4752 {
4753     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4754     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4755 }
4756 \dim_set:Nn \l_tmpb_dim
4757 {
4758     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4759     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4760 }

```

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.

```

4761 \dim_gadd:Nn \l_@@_x_initial_dim
4762 {
4763     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4764     \dim_ratio:nn
4765     {
4766         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int

```

```

4767         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4768     }
4769     { 2 \l_@@_l_dim }
4770 }
4771 \dim_gadd:Nn \l_@@_y_initial_dim
4772 {
4773     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4774     \dim_ratio:nn
4775     {
4776         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4777         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4778     }
4779     { 2 \l_@@_l_dim }
4780 }
4781 \pgf@relevantforpicturesizefalse
4782 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4783 {
4784     \pgfpathcircle
4785     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4786     { \l_@@_xdots_radius_dim }
4787     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4788     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4789 }
4790 \pgfusepathqfill
4791 }

4792 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4793 {
4794     \pgfscope
4795     \pgftransformshift
4796     {
4797         \pgfpointlineattime { 0.5 }
4798         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4799         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4800     }
4801     \fp_set:Nn \l_tmpa_fp
4802     {
4803         atand
4804         (
4805             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4806             \l_@@_x_final_dim - \l_@@_x_initial_dim
4807         )
4808     }
4809     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4810     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4811     \tl_if_empty:NF \l_@@_xdots_middle_tl
4812     {
4813         \begin { pgfscope }
4814         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4815         \pgfnode
4816         { rectangle }
4817         { center }
4818         {
4819             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4820             {
4821                 \c_math_toggle_token
4822                 \scriptstyle \l_@@_xdots_middle_tl
4823                 \c_math_toggle_token
4824             }
4825         }
4826         { }
4827         {
4828             \pgfsetfillcolor { white }

```

```

4829         \pgfusepath { fill }
4830     }
4831     \end { pgfscope }
4832 }
4833 \tl_if_empty:NF \l_@@_xdots_up_tl
4834 {
4835     \pgfnode
4836     { rectangle }
4837     { south }
4838     {
4839         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4840         {
4841             \c_math_toggle_token
4842             \scriptstyle \l_@@_xdots_up_tl
4843             \c_math_toggle_token
4844         }
4845     }
4846     { }
4847     { \pgfusepath { } }
4848 }
4849 \tl_if_empty:NF \l_@@_xdots_down_tl
4850 {
4851     \pgfnode
4852     { rectangle }
4853     { north }
4854     {
4855         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4856         {
4857             \c_math_toggle_token
4858             \scriptstyle \l_@@_xdots_down_tl
4859             \c_math_toggle_token
4860         }
4861     }
4862     { }
4863     { \pgfusepath { } }
4864 }
4865 \endpgfscope
4866 }

```

## 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.

```

4867 \hook_gput_code:nnn { begindocument } { . }
4868 {
4869     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4870     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4871     \cs_new_protected:Npn \@@_Ldots
4872     { \@@_collect_options:n { \@@_Ldots_i } }
4873     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4874     {
4875         \int_if_zero:nTF \c@jCol

```

```

4876 { \@@_error:nn { in~first~col } \Ldots }
4877 {
4878   \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4879   { \@@_error:nn { in~last~col } \Ldots }
4880   {
4881     \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4882     { #1 , down = #2 , up = #3 , middle = #4 }
4883   }
4884 }
4885 \bool_if:NF \l_@@_nullify_dots_bool
4886 { \phantom { \ensuremath { \@@_old_ldots } } }
4887 \bool_gset_true:N \g_@@_empty_cell_bool
4888 }

4889 \cs_new_protected:Npn \@@_Cdots
4890 { \@@_collect_options:n { \@@_Cdots_i } }
4891 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4892 {
4893   \int_if_zero:nTF \c@jCol
4894   { \@@_error:nn { in~first~col } \Cdots }
4895   {
4896     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4897     { \@@_error:nn { in~last~col } \Cdots }
4898     {
4899       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4900       { #1 , down = #2 , up = #3 , middle = #4 }
4901     }
4902   }
4903   \bool_if:NF \l_@@_nullify_dots_bool
4904   { \phantom { \ensuremath { \@@_old_cdots } } }
4905   \bool_gset_true:N \g_@@_empty_cell_bool
4906 }

4907 \cs_new_protected:Npn \@@_Vdots
4908 { \@@_collect_options:n { \@@_Vdots_i } }
4909 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4910 {
4911   \int_if_zero:nTF \c@iRow
4912   { \@@_error:nn { in~first~row } \Vdots }
4913   {
4914     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4915     { \@@_error:nn { in~last~row } \Vdots }
4916     {
4917       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4918       { #1 , down = #2 , up = #3 , middle = #4 }
4919     }
4920   }
4921   \bool_if:NF \l_@@_nullify_dots_bool
4922   { \phantom { \ensuremath { \@@_old_vdots } } }
4923   \bool_gset_true:N \g_@@_empty_cell_bool
4924 }

4925 \cs_new_protected:Npn \@@_Ddots
4926 { \@@_collect_options:n { \@@_Ddots_i } }
4927 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4928 {
4929   \int_case:nnF \c@iRow
4930   {
4931     0 { \@@_error:nn { in~first~row } \Ddots }
4932     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4933   }

```

```

4934 {
4935   \int_case:nnF \c@jCol
4936   {
4937     0 { \@@_error:nn { in~first~col } \Ddots }
4938     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4939   }
4940   {
4941     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4942     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4943     { #1 , down = #2 , up = #3 , middle = #4 }
4944   }
4945 }
4946 }
4947 \bool_if:NF \l_@@_nullify_dots_bool
4948 { \phantom { \ensuremath { \@@_old_ddots } } }
4949 \bool_gset_true:N \g_@@_empty_cell_bool
4950 }

4951 \cs_new_protected:Npn \@@_Iddots
4952 { \@@_collect_options:n { \@@_Iddots_i } }
4953 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4954 {
4955   \int_case:nnF \c@iRow
4956   {
4957     0 { \@@_error:nn { in~first~row } \Iddots }
4958     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4959   }
4960   {
4961     \int_case:nnF \c@jCol
4962     {
4963       0 { \@@_error:nn { in~first~col } \Iddots }
4964       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4965     }
4966     {
4967       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4968       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4969       { #1 , down = #2 , up = #3 , middle = #4 }
4970     }
4971   }
4972   \bool_if:NF \l_@@_nullify_dots_bool
4973   { \phantom { \ensuremath { \@@_old_iddots } } }
4974   \bool_gset_true:N \g_@@_empty_cell_bool
4975 }
4976 }

```

End of the \AddToHook.

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

```

4977 \keys_define:nn { NiceMatrix / Ddots }
4978 {
4979   draw-first .bool_set:N = \l_@@_draw_first_bool ,
4980   draw-first .default:n = true ,
4981   draw-first .value_forbidden:n = true
4982 }

```

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

```

4983 \cs_new_protected:Npn \@@_Hspace:
4984 {
4985   \bool_gset_true:N \g_@@_empty_cell_bool
4986   \hspace
4987 }

```

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`.

```
4988 \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`.

```
4989 \cs_new:Npn \@@_Hdotsfor:
4990 {
4991   \bool_lazy_and:nnTF
4992     { \int_if_zero_p:n \c@jCol }
4993     { \int_if_zero_p:n \l_@@_first_col_int }
4994     {
4995       \bool_if:NTF \g_@@_after_col_zero_bool
4996       {
4997         \multicolumn { 1 } { c } { }
4998         \@@_Hdotsfor_i
4999       }
5000       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5001     }
5002   {
5003     \multicolumn { 1 } { c } { }
5004     \@@_Hdotsfor_i
5005   }
5006 }
```

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:`).

```
5007 \hook_gput_code:nnn { begindocument } { . }
5008 {
5009   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5010   \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.

```
5011 \cs_new_protected:Npn \@@_Hdotsfor_i
5012 { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5013 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5014 {
5015   \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5016   {
5017     \@@_Hdotsfor:nnnn
5018     { \int_use:N \c@iRow }
5019     { \int_use:N \c@jCol }
5020     { #2 }
5021     {
5022       #1 , #3 ,
5023       down = \exp_not:n { #4 } ,
5024       up = \exp_not:n { #5 } ,
5025       middle = \exp_not:n { #6 }
5026     }
5027   }
5028   \prg_replicate:nn { #2 - 1 }
5029   {
5030     &
5031     \multicolumn { 1 } { c } { }
5032     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5033   }
5034 }
5035 }
```

```

5036 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5037 {
5038   \bool_set_false:N \l_@@_initial_open_bool
5039   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5040   \int_set:Nn \l_@@_initial_i_int { #1 }
5041   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5042   \int_compare:nNnTF { #2 } = \c_one_int
5043   {
5044     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5045     \bool_set_true:N \l_@@_initial_open_bool
5046   }
5047   {
5048     \cs_if_exist:cTF
5049     {
5050       pgf @ sh @ ns @ \@@_env:
5051       - \int_use:N \l_@@_initial_i_int
5052       - \int_eval:n { #2 - 1 }
5053     }
5054     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5055     {
5056       \int_set:Nn \l_@@_initial_j_int { #2 }
5057       \bool_set_true:N \l_@@_initial_open_bool
5058     }
5059   }
5060   \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5061   {
5062     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5063     \bool_set_true:N \l_@@_final_open_bool
5064   }
5065   {
5066     \cs_if_exist:cTF
5067     {
5068       pgf @ sh @ ns @ \@@_env:
5069       - \int_use:N \l_@@_final_i_int
5070       - \int_eval:n { #2 + #3 }
5071     }
5072     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5073     {
5074       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5075       \bool_set_true:N \l_@@_final_open_bool
5076     }
5077   }
5078   \group_begin:
5079   \@@_open_shorten:
5080   \int_if_zero:nTF { #1 }
5081   { \color { nicematrix-first-row } }
5082   {
5083     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5084     { \color { nicematrix-last-row } }
5085   }
5086
5087   \keys_set:nn { NiceMatrix / xdots } { #4 }
5088   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5089   \@@_actually_draw_Ldots:
5090   \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`).

```

5091 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5092 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5093 }

5094 \hook_gput_code:nnn { begindocument } { . }
5095 {
5096   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5097   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5098   \cs_new_protected:Npn \@@_Vdotsfor:
5099     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5100   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5101     {
5102       \bool_gset_true:N \g_@@_empty_cell_bool
5103       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5104         {
5105           \@@_Vdotsfor:nnnn
5106           { \int_use:N \c@iRow }
5107           { \int_use:N \c@jCol }
5108           { #2 }
5109           {
5110             #1 , #3 ,
5111             down = \exp_not:n { #4 } ,
5112             up = \exp_not:n { #5 } ,
5113             middle = \exp_not:n { #6 }
5114           }
5115         }
5116     }
5117 }

```

```

5118 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5119 {
5120   \bool_set_false:N \l_@@_initial_open_bool
5121   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5122 \int_set:Nn \l_@@_initial_j_int { #2 }
5123 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5124 \int_compare:nNnTF { #1 } = \c_one_int
5125 {
5126   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5127   \bool_set_true:N \l_@@_initial_open_bool
5128 }
5129 {
5130   \cs_if_exist:cTF
5131     {
5132       pgf @ sh @ ns @ \@@_env:
5133       - \int_eval:n { #1 - 1 }
5134       - \int_use:N \l_@@_initial_j_int
5135     }
5136     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5137     {
5138       \int_set:Nn \l_@@_initial_i_int { #1 }
5139       \bool_set_true:N \l_@@_initial_open_bool
5140     }
5141 }
5142 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5143 {
5144   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5145   \bool_set_true:N \l_@@_final_open_bool
5146 }
5147 {

```

```

5148 \cs_if_exist:cTF
5149 {
5150     pgf @ sh @ ns @ \@@_env:
5151     - \int_eval:n { #1 + #3 }
5152     - \int_use:N \l_@@_final_j_int
5153 }
5154 { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5155 {
5156     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5157     \bool_set_true:N \l_@@_final_open_bool
5158 }
5159 }

5160 \group_begin:
5161 \@@_open_shorten:
5162 \int_if_zero:nTF { #2 }
5163 { \color { nicematrix-first-col } }
5164 {
5165     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5166     { \color { nicematrix-last-col } }
5167 }
5168 \keys_set:nn { NiceMatrix / xdots } { #4 }
5169 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5170 \@@_actually_draw_Vdots:
5171 \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).

```

5172 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5173 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5174 }

```

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

```

5175 \NewDocumentCommand \@@_rotate: { 0 { } }
5176 {
5177     \peek_remove_spaces:n
5178     {
5179         \bool_gset_true:N \g_@@_rotate_bool
5180         \keys_set:nn { NiceMatrix / rotate } { #1 }
5181     }
5182 }

5183 \keys_define:nn { NiceMatrix / rotate }
5184 {
5185     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5186     c .value_forbidden:n = true ,
5187     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5188 }

```

## 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).<sup>13</sup>

```

5189 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5190 {
5191   \tl_if_empty:nTF { #2 }
5192     { #1 }
5193     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5194 }
5195 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5196 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

5197 \hook_gput_code:nnn { begindocument } { . }
5198 {
5199   \cs_set_nopar:Npn \l_@@_argspec_tl
5200     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5201   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5202   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5203     {
5204       \group_begin:
5205       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5206       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5207       \use:e
5208       {
5209         \@@_line_i:nn
5210         { \@@_double_int_eval:n #2 - \q_stop }
5211         { \@@_double_int_eval:n #3 - \q_stop }
5212       }
5213       \group_end:
5214     }
5215 }
5216 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5217 {
5218   \bool_set_false:N \l_@@_initial_open_bool
5219   \bool_set_false:N \l_@@_final_open_bool
5220   \bool_lazy_or:nnTF
5221     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5222     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5223     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

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

```

5224   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5225 }
5226 \hook_gput_code:nnn { begindocument } { . }
5227 {
5228   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5229   {

```

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:`.

```

5230   \c_@@_pgfortikzpicture_tl
5231   \@@_draw_line_iii:nn { #1 } { #2 }

```

---

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

```

5232     \c_@@_endpgfortikzpicture_tl
5233   }
5234 }

```

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

```

5235 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5236 {
5237   \pgfrememberpicturepositiononpagetrue
5238   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5239   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5240   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5241   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5242   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5243   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5244   \@@_draw_line:
5245 }

```

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

## 21 The command `\RowStyle`

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

```
\@@_if_row_less_than:nn { number } { instructions }
```

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

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

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

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

```

5246 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5247 { \int_compare:nNnT \c@iRow < { #1 } { #2 } }

```

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

```

5248 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5249 {
5250   \tl_gput_right:Nx \g_@@_row_style_tl
5251   {

```

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

```

5252     \exp_not:N
5253     \@@_if_row_less_than:nn
5254     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5255     { \exp_not:n { #1 } }
5256   }
5257 }
5258 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5259 \keys_define:nn { NiceMatrix / RowStyle }
5260 {
5261   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5262   cell-space-top-limit .value_required:n = true ,
5263   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5264   cell-space-bottom-limit .value_required:n = true ,

```

```

5265 cell-space-limits .meta:n =
5266 {
5267     cell-space-top-limit = #1 ,
5268     cell-space-bottom-limit = #1 ,
5269 } ,
5270 color .tl_set:N = \l_@@_color_tl ,
5271 color .value_required:n = true ,
5272 bold .bool_set:N = \l_@@_bold_row_style_bool ,
5273 bold .default:n = true ,
5274 nb-rows .code:n =
5275     \str_if_eq:nnTF { #1 } { * }
5276     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5277     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5278 nb-rows .value_required:n = true ,
5279 rowcolor .tl_set:N = \l_tmpa_tl ,
5280 rowcolor .value_required:n = true ,
5281 rowcolor .initial:n = ,
5282 unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5283 }

```

```

5284 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5285 {
5286     \group_begin:
5287     \tl_clear:N \l_tmpa_tl % value of \rowcolor
5288     \tl_clear:N \l_@@_color_tl
5289     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5290     \dim_zero:N \l_tmpa_dim
5291     \dim_zero:N \l_tmpb_dim
5292     \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5293     \tl_if_empty:NF \l_tmpa_tl
5294     {

```

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

```

5295         \tl_gput_right:Nx \g_@@_pre_code_before_tl
5296         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5297             \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5298             { \int_use:N \c@iRow - \int_use:N \c@jCol }
5299             { \int_use:N \c@iRow - * }
5300         }

```

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

```

5301         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5302         {
5303             \tl_gput_right:Nx \g_@@_pre_code_before_tl
5304             {
5305                 \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5306                 {
5307                     \int_eval:n { \c@iRow + 1 }
5308                     - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5309                 }
5310             }
5311         }
5312     }
5313     \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5314     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5315     {
5316         \exp_args:Nx \@@_put_in_row_style:n
5317         {
5318             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5319             {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5320         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5321         { \dim_use:N \l_tmpa_dim }
5322     }
5323 }
5324 }

```

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

```

5325     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5326     {
5327         \exp_args:Nx \@@_put_in_row_style:n
5328         {
5329             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5330             {
5331                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5332                 { \dim_use:N \l_tmpb_dim }
5333             }
5334         }
5335     }

```

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

```

5336     \tl_if_empty:NF \l_@@_color_tl
5337     {
5338         \@@_put_in_row_style:e
5339         {
5340             \mode_leave_vertical:
5341             \@@_color:n { \l_@@_color_tl }
5342         }
5343     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5344     \bool_if:NT \l_@@_bold_row_style_bool
5345     {
5346         \@@_put_in_row_style:n
5347         {
5348             \exp_not:n
5349             {
5350                 \if_mode_math:
5351                     \c_math_toggle_token
5352                     \bfseries \boldmath
5353                     \c_math_toggle_token
5354                 \else:
5355                     \bfseries \boldmath
5356                 \fi:
5357             }
5358         }
5359     }
5360     \group_end:
5361     \g_@@_row_style_tl
5362     \ignorespaces
5363 }

```

## 22 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That's why we try to draw rectangles of the same color in the same instruction `\pgfusepath { fill }` (and they will be in the same instruction `fill`—coded `f`—in the resulting PDF).

The commands `\@@_rowcolor`, `\@@_columncolor`, `\@@_rectanglecolor` and `\@@_rowlistcolors` don't directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence `\g_@@_colors_seq` will be built containing all the colors used by at least one of these instructions. Each *color* may be prefixed by its color model (eg: `[gray]{0.5}`).
- For the color whose index in `\g_@@_colors_seq` is equal to *i*, a list of instructions which use that color will be constructed in the token list `\g_@@_color_i_tl`. In that token list, the instructions will be written using `\@@_cartesian_color:nn` and `\@@_rectanglecolor:nn`.

#1 is the color and #2 is an instruction using that color. Despite its name, the command `\@@_add_to_colors_seq:nn` doesn't only add a color to `\g_@@_colors_seq`: it also updates the corresponding token list `\g_@@_color_i_tl`. We add in a global way because the final user may use the instructions such as `\cellcolor` in a loop of `pgffor` in the `\CodeBefore` (and we recall that a loop of `pgffor` is encapsulated in a group).

```
5364 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5365 {
```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```
5366 \int_zero:N \l_tmpa_int
```

We don't take into account the colors like `myserie!!+` because those colors are special color from a `\definecolorseries` of `xcolor`.

```
5367 \str_if_in:nnF { #1 } { !! }
5368 {
5369 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5370 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5371 }
5372 \int_if_zero:nTF \l_tmpa_int
```

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

```
5373 {
5374 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5375 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5376 }
```

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

```
5377 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5378 }

5379 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5380 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5381 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5382 {
5383 \dim_compare:nNt \l_@@_tab_rounded_corners_dim > \c_zero_dim
5384 {
```

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

```
5385 \group_begin:
5386 \pgfsetcornersarced
5387 {
5388 \pgfpoint
5389 { \l_@@_tab_rounded_corners_dim }
5390 { \l_@@_tab_rounded_corners_dim }
5391 }
```

Because we want `nicematrix` compatible with arrays constructed by `array`, the nodes for the rows and columns (that is to say the nodes `row-i` and `col-j`) have not always the expected position, that is to say, there is sometimes a slight shifting of something such as `\arrayrulewidth`. Now, for the clipping, we have to change slightly the position of that clipping whether a rounded rectangle around the array is required. That's the point which is tested in the following line.

```

5392     \bool_if:NTF \l_@@_hvlines_bool
5393     {
5394         \pgfpathrectanglecorners
5395         {
5396             \pgfpointadd
5397             { \@@_qpoint:n { row-1 } }
5398             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5399         }
5400         {
5401             \pgfpointadd
5402             {
5403                 \@@_qpoint:n
5404                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5405             }
5406             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5407         }
5408     }
5409     {
5410         \pgfpathrectanglecorners
5411         { \@@_qpoint:n { row-1 } }
5412         {
5413             \pgfpointadd
5414             {
5415                 \@@_qpoint:n
5416                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5417             }
5418             { \pgfpoint \c_zero_dim \arrayrulewidth }
5419         }
5420     }
5421     \pgfusepath { clip }
5422     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5423 }
5424 }

```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```

5425 \cs_new_protected:Npn \@@_actually_color:
5426 {
5427     \pgfpicture
5428     \pgf@relevantforpicturesizefalse

```

If the final user has used the key `rounded-corners` for the environment `{NiceTabular}`, we will clip to a rectangle with rounded corners before filling the rectangles.

```

5429     \@@_clip_with_rounded_corners:
5430     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5431     {
5432         \int_compare:nNnTF { ##1 } = \c_one_int
5433         {
5434             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5435             \use:c { g_@@_color _ 1 _tl }
5436             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5437         }
5438         {
5439             \begin { pgfscope }
5440                 \@@_color_opacity ##2
5441                 \use:c { g_@@_color _ ##1 _tl }

```

```

5442         \tl_gclear:c { g_@@_color _ ##1 _tl }
5443         \pgfusepath { fill }
5444     \end { pgfscope }
5445 }
5446 }
5447 \endpgfpicture
5448 }

```

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

```

5449 \cs_new_protected:Npn \@@_color_opacity
5450 {
5451     \peek_meaning:NTF [
5452         { \@@_color_opacity:w }
5453         { \@@_color_opacity:w [ ] }
5454     }

```

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

```

5455 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5456 {
5457     \tl_clear:N \l_tmpa_tl
5458     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5459     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5460     \tl_if_empty:NTF \l_tmpb_tl
5461         { \@declaredcolor }
5462         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5463 }

```

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

```

5464 \keys_define:nn { nicematrix / color-opacity }
5465 {
5466     opacity .tl_set:N          = \l_tmpa_tl ,
5467     opacity .value_required:n = true
5468 }

```

```

5469 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5470 {
5471     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5472     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5473     \@@_cartesian_path:
5474 }

```

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

```

5475 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5476 {
5477     \tl_if_blank:nF { #2 }
5478     {
5479         \@@_add_to_colors_seq:en
5480         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5481         { \@@_cartesian_color:nn { #3 } { - } }
5482     }
5483 }

```

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

```

5484 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5485 {
5486   \tl_if_blank:nF { #2 }
5487   {
5488     \@@_add_to_colors_seq:en
5489     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5490     { \@@_cartesian_color:nn { - } { #3 } }
5491   }
5492 }

```

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

```

5493 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5494 {
5495   \tl_if_blank:nF { #2 }
5496   {
5497     \@@_add_to_colors_seq:en
5498     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5499     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5500   }
5501 }

```

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

```

5502 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5503 {
5504   \tl_if_blank:nF { #2 }
5505   {
5506     \@@_add_to_colors_seq:en
5507     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5508     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5509   }
5510 }

```

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

```

5511 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5512 {
5513   \@@_cut_on_hyphen:w #1 \q_stop
5514   \tl_clear_new:N \l_@@_tmpc_tl
5515   \tl_clear_new:N \l_@@_tmpd_tl
5516   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5517   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5518   \@@_cut_on_hyphen:w #2 \q_stop
5519   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5520   \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5521   \@@_cartesian_path:n { #3 }
5522 }

```

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

```

5523 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5524 {
5525   \clist_map_inline:nn { #3 }
5526   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5527 }

```

```

5528 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5529 {
5530   \int_step_inline:nn \c@iRow

```

```

5531 {
5532   \int_step_inline:nn \c@jCol
5533   {
5534     \int_if_even:nTF { #####1 + #1 }
5535     { \@@_cellcolor [ #1 ] { #2 } }
5536     { \@@_cellcolor [ #1 ] { #3 } }
5537     { #1 - #####1 }
5538   }
5539 }
5540 }

```

The command `\@@_arraycolor` (linked to `\arraycolor` at the beginning of the `\CodeBefore`) will color the whole tabular (excepted the potential exterior rows and columns) and the cells in the “corners”.

```

5541 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5542 {
5543   \@@_rectanglecolor [ #1 ] { #2 }
5544   { 1 - 1 }
5545   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5546 }

```

```

5547 \keys_define:nn { NiceMatrix / rowcolors }
5548 {
5549   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5550   respect-blocks .default:n = true ,
5551   cols .tl_set:N = \l_@@_cols_tl ,
5552   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5553   restart .default:n = true ,
5554   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5555 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

**#1** (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```

5556 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5557 {

```

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

```

5558   \group_begin:
5559   \seq_clear_new:N \l_@@_colors_seq
5560   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5561   \tl_clear_new:N \l_@@_cols_tl
5562   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5563   \keys_set:nn { NiceMatrix / rowcolors } { #4 }

```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```

5564   \int_zero_new:N \l_@@_color_int
5565   \int_set_eq:NN \l_@@_color_int \c_one_int
5566   \bool_if:NT \l_@@_respect_blocks_bool
5567   {

```

We don’t want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that’s why we filter the sequence of the blocks (in a the sequence `\l_tmpa_seq`).

```

5568     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5569     \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5570     { \@@_not_in_exterior_p:nnnnn ##1 }
5571   }

```

```

5572 \pgfpicture
5573 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5574 \clist_map_inline:nn { #2 }
5575 {
5576   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5577   \tl_if_in:NnTF \l_tmpa_tl { - }
5578   { \@@_cut_on_hyphen:w ##1 \q_stop }
5579   { \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.

```

5580   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5581   \int_set:Nn \l_@@_color_int
5582   { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5583   \int_zero_new:N \l_@@_tmpc_int
5584   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5585   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5586   {

```

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

```

5587   \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

5588   \bool_if:NT \l_@@_respect_blocks_bool
5589   {
5590     \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5591     { \@@_intersect_our_row_p:nnnnn ###1 }
5592     \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5593   }
5594   \tl_set:No \l_@@_rows_tl
5595   { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5596   \tl_clear_new:N \l_@@_color_tl
5597   \tl_set:Nx \l_@@_color_tl
5598   {
5599     \@@_color_index:n
5600     {
5601       \int_mod:nn
5602       { \l_@@_color_int - 1 }
5603       { \seq_count:N \l_@@_colors_seq }
5604       + 1
5605     }
5606   }
5607   \tl_if_empty:NF \l_@@_color_tl
5608   {
5609     \@@_add_to_colors_seq:ee
5610     { \tl_if_blank:NF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5611     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5612   }
5613   \int_incr:N \l_@@_color_int
5614   \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5615 }
5616 }
5617 \endpgfpicture
5618 \group_end:
5619 }

```

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

```

5620 \cs_new:Npn \@@_color_index:n #1
5621 {

```

```

5622 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5623 { \@@_color_index:n { #1 - 1 } }
5624 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5625 }

```

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

```

5626 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5627 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

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

```

5628 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5629 {
5630   \int_compare:nNt { #3 } > \l_tmpb_int
5631   { \int_set:Nn \l_tmpb_int { #3 } }
5632 }

```

```

5633 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5634 {
5635   \int_if_zero:nTF { #4 }
5636   \prg_return_false:
5637   {
5638     \int_compare:nNtTF { #2 } > \c@jCol
5639     \prg_return_false:
5640     \prg_return_true:
5641   }
5642 }

```

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

```

5643 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5644 {
5645   \int_compare:nNtTF { #1 } > \l_tmpa_int
5646   \prg_return_false:
5647   {
5648     \int_compare:nNtTF \l_tmpa_int > { #3 }
5649     \prg_return_false:
5650     \prg_return_true:
5651   }
5652 }

```

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`).

```

5653 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5654 {
5655   \dim_compare:nNtTF { #1 } = \c_zero_dim
5656   {
5657     \bool_if:NTF
5658     \@@_nocolor_used_bool
5659     \@@_cartesian_path_normal_ii:
5660     {
5661       \seq_if_empty:NTF \l_@@_corners_cells_seq
5662       { \@@_cartesian_path_normal_i:n { #1 } }
5663       \@@_cartesian_path_normal_ii:
5664     }

```

```

5665     }
5666     { \@@_cartesian_path_normal_i:n { #1 } }
5667 }

```

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.

```

5668 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5669 {
5670   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5671   \clist_map_inline:Nn \l_@@_cols_tl
5672   {
5673     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5674     \tl_if_in:NnTF \l_tmpa_tl { - }
5675     { \@@_cut_on_hyphen:w ##1 \q_stop }
5676     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5677     \tl_if_empty:NNTF \l_tmpa_tl
5678     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5679     {
5680       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5681       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5682     }
5683     \tl_if_empty:NNTF \l_tmpb_tl
5684     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5685     {
5686       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5687       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5688     }
5689     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5690     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5691     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5692     \@@_qpoint:n { col - \l_tmpa_tl }
5693     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5694     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5695     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5696     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5697     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5698   \clist_map_inline:Nn \l_@@_rows_tl
5699   {
5700     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5701     \tl_if_in:NnTF \l_tmpa_tl { - }
5702     { \@@_cut_on_hyphen:w #####1 \q_stop }
5703     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5704     \tl_if_empty:NNTF \l_tmpa_tl
5705     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5706     {
5707       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5708       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5709     }
5710     \tl_if_empty:NNTF \l_tmpb_tl
5711     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5712     {
5713       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5714       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5715     }
5716     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5717     { \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`.

```

5718         \cs_if_exist:cF
5719         { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5720         {
5721             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5722             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5723             \@@_qpoint:n { row - \l_tmpa_tl }
5724             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5725             \pgfpathrectanglecorners
5726             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5727             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5728         }
5729     }
5730 }
5731 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5732 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5733 {
5734     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5735     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5736     \clist_map_inline:Nn \l_@@_cols_tl
5737     {
5738         \@@_qpoint:n { col - ##1 }
5739         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5740         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5741         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5742         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5743         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5744     \clist_map_inline:Nn \l_@@_rows_tl
5745     {
5746         \seq_if_in:NnF \l_@@_corners_cells_seq
5747         { #####1 - ##1 }
5748         {
5749             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5750             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5751             \@@_qpoint:n { row - #####1 }
5752             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5753             \cs_if_exist:cF
5754             { @@ _ #####1 _ ##1 _ nocolor }
5755             {
5756                 \pgfpathrectanglecorners
5757                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5758                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5759             }
5760         }
5761     }
5762 }
5763 }

```

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`).

```

5764 \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

```

5765 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1

```

```

5766 {
5767   \bool_set_true:N \@@_nocolor_used_bool
5768   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5769   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5770   \clist_map_inline:Nn \l_@@_rows_tl
5771   {
5772     \clist_map_inline:Nn \l_@@_cols_tl
5773     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5774   }
5775 }

```

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

```

5776 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5777 {
5778   \clist_set_eq:NN \l_tmpa_clist #1
5779   \clist_clear:N #1
5780   \clist_map_inline:Nn \l_tmpa_clist
5781   {
5782     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5783     \tl_if_in:NnTF \l_tmpa_tl { - }
5784     { \@@_cut_on_hyphen:w ##1 \q_stop }
5785     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5786     \bool_lazy_or:nnT
5787     { \tl_if_blank_p:o \l_tmpa_tl }
5788     { \str_if_eq_p:on \l_tmpa_tl { * } }
5789     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5790     \bool_lazy_or:nnT
5791     { \tl_if_blank_p:o \l_tmpb_tl }
5792     { \str_if_eq_p:on \l_tmpb_tl { * } }
5793     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5794     \int_compare:nNnT \l_tmpb_tl > #2
5795     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5796     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5797     { \clist_put_right:Nn #1 { #####1 } }
5798   }
5799 }

```

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

```

5800 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5801 {
5802   \@@_test_color_inside:
5803   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5804   {

```

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`).

```

5805     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5806     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5807   }
5808   \ignorespaces
5809 }

```

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

```

5810 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5811 {
5812   \@@_test_color_inside:

```

```

5813 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5814 {
5815   \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5816   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5817   { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5818 }
5819 \ignorespaces
5820 }

```

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 currying).

```

5821 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5822 { \@@_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.

```

5823 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5824 {
5825   \@@_test_color_inside:
5826   \peek_remove_spaces:n
5827   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5828 }

```

```

5829 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5830 {

```

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`.

```

5831 \seq_gclear:N \g_tmpa_seq
5832 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5833 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5834 \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`.

```

5835 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5836 {
5837   { \int_use:N \c@iRow }
5838   { \exp_not:n { #1 } }
5839   { \exp_not:n { #2 } }
5840   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5841 }
5842 }

```

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`).

```

5843 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5844 {
5845   \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).

```

5846     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5847     {
5848       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5849       {
5850         \@@_rowlistcolors
5851         [ \exp_not:n { #2 } ]
5852         { #1 - \int_eval:n { \c@iRow - 1 } }
5853         { \exp_not:n { #3 } }
5854         [ \exp_not:n { #4 } ]
5855       }
5856     }
5857 }

```

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.

```

5858 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5859 {
5860   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5861   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5862   \seq_gclear:N \g_@@_rowlistcolors_seq
5863 }

5864 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5865 {
5866   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5867   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5868 }

```

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.

```

5869 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5870 {

```

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).

```

5871   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5872   {

```

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).

```

5873     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5874     {
5875       \exp_not:N \columncolor [ #1 ]
5876       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5877     }
5878   }
5879 }

```

```

5880 \hook_gput_code:nnn { begindocument } { . }
5881 {
5882   \IfPackageLoadedTF { colortbl }
5883   {
5884     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5885     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5886     \cs_new_protected:Npn \@@_revert_colortbl:

```

```

5887     {
5888         \hook_gput_code:nnn { env / tabular / begin } { . }
5889         {
5890             \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5891             \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5892         }
5893     }
5894 }
5895 { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5896 }

```

## 23 The vertical and horizontal rules

### OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` 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`.

```

5897 \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`.

```

5898 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5899 {
5900     \int_if_zero:nTF \l_@@_first_col_int
5901     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5902     {
5903         \int_if_zero:nTF \c@jCol
5904         {
5905             \int_compare:nNnF \c@iRow = { -1 }
5906             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5907         }
5908         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5909     }
5910 }

```

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.

```

5911 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5912 {
5913     \int_if_zero:nF \c@iRow
5914     {
5915         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5916         {
5917             \int_compare:nNnT \c@jCol > \c_zero_int
5918             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5919         }
5920     }
5921 }

```

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.

```

5922 \keys_define:nn { NiceMatrix / Rules }
5923 {
5924   position .int_set:N = \l_@@_position_int ,
5925   position .value_required:n = true ,
5926   start .int_set:N = \l_@@_start_int ,
5927   end .code:n =
5928     \bool_lazy_or:nnTF
5929     { \tl_if_empty_p:n { #1 } }
5930     { \str_if_eq_p:nn { #1 } { last } }
5931     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5932     { \int_set:Nn \l_@@_end_int { #1 } }
5933 }

```

It’s possible that the rule won’t be drawn continuously from `start` ot `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.

```

5934 \keys_define:nn { NiceMatrix / RulesBis }
5935 {
5936   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5937   multiplicity .initial:n = 1 ,
5938   dotted .bool_set:N = \l_@@_dotted_bool ,
5939   dotted .initial:n = false ,
5940   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]`.

```

5941   color .code:n =
5942     \@@_set_CT@arc@:n { #1 }
5943     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5944   color .value_required:n = true ,
5945   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5946   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.

```

5947   tikz .code:n =
5948     \IfPackageLoadedTF { tikz }
5949     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5950     { \@@_error:n { tikz-without-tikz } } ,
5951   tikz .value_required:n = true ,
5952   total-width .dim_set:N = \l_@@_rule_width_dim ,
5953   total-width .value_required:n = true ,
5954   width .meta:n = { total-width = #1 } ,
5955   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5956 }

```

## The vertical rules

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

```

5957 \cs_new_protected:Npn \@@_vline:n #1
5958 {

```

The group is for the options.

```

5959 \group_begin:
5960 \int_set_eq:NN \l_@@_end_int \c@iRow
5961 \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).

```

5962 \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5963 \@@_vline_i:
5964 \group_end:
5965 }

5966 \cs_new_protected:Npn \@@_vline_i:
5967 {

```

$\l\_tmpa\_tl$  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  $\l\_@@\_tmpc\_tl$ .

```

5968 \tl_set:No \l\_tmpb\_tl { \int_use:N \l_@@_position_int }
5969 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5970 \l\_tmpa\_tl
5971 {

```

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  $\backslash\text{Block}$  or a virtual block corresponding to a dotted line, created by  $\backslash\text{Cdots}$ ,  $\backslash\text{Vdots}$ , etc.), we will set  $\g\_tmpa\_bool$  to false and the small vertical rule won't be drawn.

```

5972 \bool_gset_true:N \g\_tmpa\_bool
5973 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5974 { \@@_test_vline_in_block:nnnnn ##1 }
5975 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5976 { \@@_test_vline_in_block:nnnnn ##1 }
5977 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5978 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5979 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5980 \bool_if:NTF \g\_tmpa\_bool
5981 {
5982 \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.

```

5983 { \int_set:Nn \l_@@_local_start_int \l\_tmpa\_tl }
5984 }
5985 {
5986 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
5987 {
5988 \int_set:Nn \l_@@_local_end_int { \l\_tmpa\_tl - 1 }
5989 \@@_vline_ii:
5990 \int_zero:N \l_@@_local_start_int
5991 }
5992 }
5993 }
5994 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
5995 {
5996 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5997 \@@_vline_ii:
5998 }
5999 }

```

```

6000 \cs_new_protected:Npn \@@_test_in_corner_v:
6001 {
6002 \int_compare:nNnTF \l\_tmpb\_tl = { \int_eval:n { \c@jCol + 1 } }
6003 {
6004 \seq_if_in:NxT
6005 \l_@@_corners_cells_seq

```

```

6006         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6007         { \bool_set_false:N \g_tmpa_bool }
6008     }
6009     {
6010         \seq_if_in:NxT
6011         \l_@@_corners_cells_seq
6012         { \l_tmpa_tl - \l_tmpb_tl }
6013         {
6014             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6015             { \bool_set_false:N \g_tmpa_bool }
6016             {
6017                 \seq_if_in:NxT
6018                 \l_@@_corners_cells_seq
6019                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6020                 { \bool_set_false:N \g_tmpa_bool }
6021             }
6022         }
6023     }
6024 }

6025 \cs_new_protected:Npn \@@_vline_ii:
6026 {
6027     \tl_clear:N \l_@@_tikz_rule_tl
6028     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6029     \bool_if:NTF \l_@@_dotted_bool
6030     \@@_vline_iv:
6031     {
6032         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6033         \@@_vline_iii:
6034         \@@_vline_v:
6035     }
6036 }

```

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

```

6037 \cs_new_protected:Npn \@@_vline_iii:
6038 {
6039     \pgfpicture
6040     \pgfrememberpicturepositiononpagetrue
6041     \pgf@relevantforpicturesizefalse
6042     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6043     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6044     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6045     \dim_set:Nn \l_tmpb_dim
6046     {
6047         \pgf@x
6048         - 0.5 \l_@@_rule_width_dim
6049         +
6050         ( \arrayrulewidth * \l_@@_multiplicity_int
6051           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6052     }
6053     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6054     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6055     \bool_lazy_all:nT
6056     {
6057         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6058         { \cs_if_exist_p:N \CT@drsc@ }
6059         { ! \tl_if_blank_p:o \CT@drsc@ }
6060     }
6061     {
6062         \group_begin:
6063         \CT@drsc@
6064         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

```

6065 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6066 \dim_set:Nn \l_@@_tmpd_dim
6067 {
6068   \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6069   * ( \l_@@_multiplicity_int - 1 )
6070 }
6071 \pgfpathrectanglecorners
6072 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6073 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6074 \pgfusepath { fill }
6075 \group_end:
6076 }
6077 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6078 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6079 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6080 {
6081   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6082   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6083   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6084   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6085 }
6086 \CT@arc@
6087 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6088 \pgfsetrectcap
6089 \pgfusepathqstroke
6090 \endpgfpicture
6091 }

```

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

```

6092 \cs_new_protected:Npn \@@_vline_iv:
6093 {
6094   \pgfpicture
6095   \pgfrememberpicturepositiononpagetrue
6096   \pgf@relevantforpicturesizefalse
6097   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6098   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6099   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6100   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6101   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6102   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6103   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6104   \CT@arc@
6105   \@@_draw_line:
6106   \endpgfpicture
6107 }

```

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

```

6108 \cs_new_protected:Npn \@@_vline_v:
6109 {
6110   \begin {tikzpicture }
6111   % added 2023/09/25

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6112 \CT@arc@
6113 \tl_if_empty:NF \l_@@_rule_color_tl
6114 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6115 \pgfrememberpicturepositiononpagetrue
6116 \pgf@relevantforpicturesizefalse
6117 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6118 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6119 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }

```

```

6120 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6121 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6122 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6123 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6124 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6125 ( \l_tmpb_dim , \l_tmpa_dim ) --
6126 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6127 \end { tikzpicture }
6128 }

```

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).

```

6129 \cs_new_protected:Npn \@@_draw_vlines:
6130 {
6131   \int_step_inline:nnn
6132   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6133   {
6134     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6135     \c@jCol
6136     { \int_eval:n { \c@jCol + 1 } } }
6137   }
6138   {
6139     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6140     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6141     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6142   }
6143 }

```

## 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}`.

```

6144 \cs_new_protected:Npn \@@_hline:n #1
6145 {

```

The group is for the options.

```

6146   \group_begin:
6147   \int_zero_new:N \l_@@_end_int
6148   \int_set_eq:NN \l_@@_end_int \c@jCol
6149   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6150   \@@_hline_i:
6151   \group_end:
6152 }
6153 \cs_new_protected:Npn \@@_hline_i:
6154 {
6155   \int_zero_new:N \l_@@_local_start_int
6156   \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`.

```

6157   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6158   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6159   \l_tmpb_tl
6160   {

```

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.

```

6161   \bool_gset_true:N \g_tmpa_bool
6162   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6163   { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6164 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6165 { \@@_test_hline_in_block:nnnnn ##1 }
6166 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6167 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6168 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6169 \bool_if:NTF \g_tmpa_bool
6170 {
6171 \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.

```

6172 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6173 }
6174 {
6175 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6176 {
6177 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6178 \@@_hline_ii:
6179 \int_zero:N \l_@@_local_start_int
6180 }
6181 }
6182 }
6183 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6184 {
6185 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6186 \@@_hline_ii:
6187 }
6188 }

```

```

6189 \cs_new_protected:Npn \@@_test_in_corner_h:
6190 {
6191 \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6192 {
6193 \seq_if_in:NxT
6194 \l_@@_corners_cells_seq
6195 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6196 { \bool_set_false:N \g_tmpa_bool }
6197 }
6198 {
6199 \seq_if_in:NxT
6200 \l_@@_corners_cells_seq
6201 { \l_tmpa_tl - \l_tmpb_tl }
6202 {
6203 \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6204 { \bool_set_false:N \g_tmpa_bool }
6205 {
6206 \seq_if_in:NxT
6207 \l_@@_corners_cells_seq
6208 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6209 { \bool_set_false:N \g_tmpa_bool }
6210 }
6211 }
6212 }
6213 }

```

```

6214 \cs_new_protected:Npn \@@_hline_ii:
6215 {
6216 \tl_clear:N \l_@@_tikz_rule_tl
6217 \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6218 \bool_if:NTF \l_@@_dotted_bool
6219 \@@_hline_iv:
6220 {

```

```

6221     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6222     \@@_hline_iii:
6223     \@@_hline_v:
6224 }
6225 }

```

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

```

6226 \cs_new_protected:Npn \@@_hline_iii:
6227 {
6228   \pgfpicture
6229   \pgfrememberpicturepositiononpagetrue
6230   \pgf@relevantforpicturesizefalse
6231   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6232   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6233   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6234   \dim_set:Nn \l_tmpb_dim
6235   {
6236     \pgf@y
6237     - 0.5 \l_@@_rule_width_dim
6238     +
6239     ( \arrayrulewidth * \l_@@_multiplicity_int
6240       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6241   }
6242   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6243   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6244   \bool_lazy_all:nT
6245   {
6246     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6247     { \cs_if_exist_p:N \CT@drsc@ }
6248     { ! \tl_if_blank_p:o \CT@drsc@ }
6249   }
6250   {
6251     \group_begin:
6252     \CT@drsc@
6253     \dim_set:Nn \l_@@_tmpd_dim
6254     {
6255       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6256       * ( \l_@@_multiplicity_int - 1 )
6257     }
6258     \pgfpathrectanglecorners
6259     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6260     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6261     \pgfusepathqfill
6262     \group_end:
6263   }
6264   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6265   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6266   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6267   {
6268     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6269     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6270     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6271     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6272   }
6273   \CT@arc@
6274   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6275   \pgfsetrectcap
6276   \pgfusepathqstroke
6277   \endpgfpicture
6278 }

```

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}

```

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

```

6279 \cs_new_protected:Npn \@@_hline_iv:
6280 {
6281   \pgfpicture
6282   \pgfrememberpicturepositiononpagetrue
6283   \pgf@relevantforpicturesizefalse
6284   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6285   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6286   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6287   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6288   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6289   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6290   {
6291     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6292     \bool_if:NF \g_@@_delims_bool
6293     { \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.

```

6294     \tl_if_eq:NnF \g_@@_left_delim_tl (
6295       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6296     )
6297     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6298     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6299     \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6300     {
6301       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6302       \bool_if:NF \g_@@_delims_bool
6303       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6304       \tl_if_eq:NnF \g_@@_right_delim_tl )
6305       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6306     }
6307     \CT@arc@
6308     \@@_draw_line:
6309     \endpgfpicture
6310   }

```

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`).

```

6311 \cs_new_protected:Npn \@@_hline_v:
6312 {
6313   \begin { tikzpicture }
6314   % added 2023/09/25

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6315   \CT@arc@

```

```

6316 \tl_if_empty:NF \l_@@_rule_color_tl
6317 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6318 \pgfrememberpicturepositiononpagetrue
6319 \pgf@relevantforpicturesizefalse
6320 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6321 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6322 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6323 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6324 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6325 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6326 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6327 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6328 ( \l_tmpa_dim , \l_tmpb_dim ) --
6329 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6330 \end { tikzpicture }
6331 }

```

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).

```

6332 \cs_new_protected:Npn \@@_draw_hlines:
6333 {
6334   \int_step_inline:nnn
6335   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6336   {
6337     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6338     \c@iRow
6339     { \int_eval:n { \c@iRow + 1 } }
6340   }
6341   {
6342     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6343     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6344     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6345   }
6346 }

```

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

```

6347 \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.

```

6348 \cs_set:Npn \@@_Hline_i:n #1
6349 {
6350   \peek_remove_spaces:n
6351   {
6352     \peek_meaning:NTF \Hline
6353     { \@@_Hline_ii:nn { #1 + 1 } }
6354     { \@@_Hline_iii:n { #1 } }
6355   }
6356 }
6357 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6358 \cs_set:Npn \@@_Hline_iii:n #1
6359 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6360 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6361 {
6362   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6363   \skip_vertical:N \l_@@_rule_width_dim
6364   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6365   {
6366     \@@_hline:n
6367     {
6368       multiplicity = #1 ,

```

```

6369         position = \int_eval:n { \c@iRow + 1 } ,
6370         total-width = \dim_use:N \l_@@_rule_width_dim ,
6371         #2
6372     }
6373 }
6374 \egroup
6375 }

```

### 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`).

```

6376 \cs_new_protected:Npn \@@_custom_line:n #1
6377 {
6378     \str_clear_new:N \l_@@_command_str
6379     \str_clear_new:N \l_@@_ccommand_str
6380     \str_clear_new:N \l_@@_letter_str
6381     \tl_clear_new:N \l_@@_other_keys_tl
6382     \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.

```

6383     \bool_lazy_all:nTF
6384     {
6385         { \str_if_empty_p:N \l_@@_letter_str }
6386         { \str_if_empty_p:N \l_@@_command_str }
6387         { \str_if_empty_p:N \l_@@_ccommand_str }
6388     }
6389     { \@@_error:n { No~letter~and~no~command } }
6390     { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6391 }

6392 \keys_define:nn { NiceMatrix / custom-line }
6393 {
6394     letter .str_set:N = \l_@@_letter_str ,
6395     letter .value_required:n = true ,
6396     command .str_set:N = \l_@@_command_str ,
6397     command .value_required:n = true ,
6398     ccommand .str_set:N = \l_@@_ccommand_str ,
6399     ccommand .value_required:n = true ,
6400 }

```

```

6401 \cs_new_protected:Npn \@@_custom_line_i:n #1
6402 {

```

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

```

6403     \bool_set_false:N \l_@@_tikz_rule_bool
6404     \bool_set_false:N \l_@@_dotted_rule_bool
6405     \bool_set_false:N \l_@@_color_bool

6406     \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6407     \bool_if:NT \l_@@_tikz_rule_bool
6408     {
6409         \IfPackageLoadedTF { tikz }
6410         { }
6411         { \@@_error:n { tikz~in~custom~line~without~tikz } }
6412     }
6413     \bool_if:NT \l_@@_color_bool
6414     { \@@_error:n { color~in~custom~line~with~tikz } }

```

```

6414     }
6415     \bool_if:NT \l_@@_dotted_rule_bool
6416     {
6417         \int_compare:nNtT \l_@@_multiplicity_int > \c_one_int
6418         { \@@_error:n { key~multiplicity~with~dotted } }
6419     }
6420     \str_if_empty:NF \l_@@_letter_str
6421     {
6422         \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6423         { \@@_error:n { Several~letters } }
6424         {
6425             \exp_args:NnV \tl_if_in:NnTF
6426             \c_@@_forbidden_letters_str \l_@@_letter_str
6427             { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6428         }

```

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.

```

6429         \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6430         { \@@_v_custom_line:n { #1 } }
6431     }
6432 }
6433 }
6434 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6435 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6436 }
6437 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6438 \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.

```

6439 \keys_define:nn { NiceMatrix / custom-line-bis }
6440 {
6441     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6442     multiplicity .initial:n = 1 ,
6443     multiplicity .value_required:n = true ,
6444     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6445     color .value_required:n = true ,
6446     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6447     tikz .value_required:n = true ,
6448     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6449     dotted .value_forbidden:n = true ,
6450     total-width .code:n = { } ,
6451     total-width .value_required:n = true ,
6452     width .code:n = { } ,
6453     width .value_required:n = true ,
6454     sep-color .code:n = { } ,
6455     sep-color .value_required:n = true ,
6456     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6457 }

```

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

```

6458 \bool_new:N \l_@@_dotted_rule_bool
6459 \bool_new:N \l_@@_tikz_rule_bool
6460 \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`.

```

6461 \keys_define:nn { NiceMatrix / custom-line-width }
6462 {
6463   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6464   multiplicity .initial:n = 1 ,
6465   multiplicity .value_required:n = true ,
6466   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6467   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6468   \bool_set_true:N \l_@@_total_width_bool ,
6469   total-width .value_required:n = true ,
6470   width .meta:n = { total-width = #1 } ,
6471   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6472 }

```

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).

```

6473 \cs_new_protected:Npn \@@_h_custom_line:n #1
6474 {

```

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).

```

6475   \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6476   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6477 }

```

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).

```

6478 \cs_new_protected:Npn \@@_c_custom_line:n #1
6479 {

```

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

```

6480   \exp_args:Nc \NewExpandableDocumentCommand
6481   { nicematrix - \l_@@_ccommand_str }
6482   { 0 { } m }
6483   {
6484     \noalign
6485     {
6486       \@@_compute_rule_width:n { #1 , ##1 }
6487       \skip_vertical:n { \l_@@_rule_width_dim }
6488       \clist_map_inline:nn
6489       { ##2 }
6490       { \@@_c_custom_line_i:nn { #1 , ##1 } { ##### } }
6491     }
6492   }
6493   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6494 }

```

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*.

```

6495 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6496 {
6497   \str_if_in:nnTF { #2 } { - }
6498   { \@@_cut_on_hyphen:w #2 \q_stop }
6499   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6500   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6501   {
6502     \@@_hline:n
6503     {
6504       #1 ,
6505       start = \l_tmpa_tl ,

```

```

6506         end = \l_tmpb_tl ,
6507         position = \int_eval:n { \c@iRow + 1 } ,
6508         total-width = \dim_use:N \l_@@_rule_width_dim
6509     }
6510 }
6511 }
6512 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6513 {
6514     \bool_set_false:N \l_@@_tikz_rule_bool
6515     \bool_set_false:N \l_@@_total_width_bool
6516     \bool_set_false:N \l_@@_dotted_rule_bool
6517     \keys_set_known:n { NiceMatrix / custom-line-width } { #1 }
6518     \bool_if:NF \l_@@_total_width_bool
6519     {
6520         \bool_if:NTF \l_@@_dotted_rule_bool
6521         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6522         {
6523             \bool_if:NF \l_@@_tikz_rule_bool
6524             {
6525                 \dim_set:Nn \l_@@_rule_width_dim
6526                 {
6527                     \arrayrulewidth * \l_@@_multiplicity_int
6528                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6529                 }
6530             }
6531         }
6532     }
6533 }
6534 \cs_new_protected:Npn \@@_v_custom_line:n #1
6535 {
6536     \@@_compute_rule_width:n { #1 }

```

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

```

6537     \tl_gput_right:Nx \g_@@_array_preamble_tl
6538     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6539     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6540     {
6541         \@@_vline:n
6542         {
6543             #1 ,
6544             position = \int_eval:n { \c@jCol + 1 } ,
6545             total-width = \dim_use:N \l_@@_rule_width_dim
6546         }
6547     }
6548     \@@_rec_preamble:n
6549 }
6550 \@@_custom_line:n
6551 { 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.

```

6552 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6553 {
6554     \int_compare:nNt \l_tmpa_tl > { #1 }
6555     {
6556         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
6557         {
6558             \int_compare:nNt \l_tmpb_tl > { #2 - 1 }

```

```

6559         {
6560             \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6561             { \bool_gset_false:N \g_tmpa_bool }
6562         }
6563     }
6564 }
6565 }

```

The same for vertical rules.

```

6566 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6567 {
6568     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6569     {
6570         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6571         {
6572             \int_compare:nNnT \l_tmpb_tl > { #2 }
6573             {
6574                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6575                 { \bool_gset_false:N \g_tmpa_bool }
6576             }
6577         }
6578     }
6579 }
6580 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6581 {
6582     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6583     {
6584         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6585         {
6586             \int_compare:nNnTF \l_tmpa_tl = { #1 }
6587             { \bool_gset_false:N \g_tmpa_bool }
6588             {
6589                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6590                 { \bool_gset_false:N \g_tmpa_bool }
6591             }
6592         }
6593     }
6594 }
6595 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6596 {
6597     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6598     {
6599         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6600         {
6601             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6602             { \bool_gset_false:N \g_tmpa_bool }
6603             {
6604                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6605                 { \bool_gset_false:N \g_tmpa_bool }
6606             }
6607         }
6608     }
6609 }

```

## 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.

```

6610 \cs_new_protected:Npn \@@_compute_corners:
6611 {

```

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.

```

6612 \seq_clear_new:N \l_@@_corners_cells_seq
6613 \clist_map_inline:Nn \l_@@_corners_clist
6614 {
6615 \str_case:nnF { ##1 }
6616 {
6617 { NW }
6618 { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6619 { NE }
6620 { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6621 { SW }
6622 { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6623 { SE }
6624 { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6625 }
6626 { \@@_error:nn { bad~corner } { ##1 } }
6627 }

```

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

```

6628 \seq_if_empty:NF \l_@@_corners_cells_seq
6629 {

```

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.

```

6630 \tl_gput_right:Nx \g_@@_aux_tl
6631 {
6632 \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6633 { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6634 }
6635 }
6636 }

```

“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.

```

6637 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6638 {

```

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.

```

6639 \bool_set_false:N \l_tmpa_bool
6640 \int_zero_new:N \l_@@_last_empty_row_int
6641 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6642 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6643 {
6644 \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6645 \bool_lazy_or:nnTF

```

```

6646     {
6647         \cs_if_exist_p:c
6648         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6649     }
6650     \l_tmpb_bool
6651     { \bool_set_true:N \l_tmpa_bool }
6652     {
6653         \bool_if:NF \l_tmpa_bool
6654         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6655     }
6656 }

```

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

```

6657     \bool_set_false:N \l_tmpa_bool
6658     \int_zero_new:N \l_@@_last_empty_column_int
6659     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6660     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6661     {
6662         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6663         \bool_lazy_or:nnTF
6664         \l_tmpb_bool
6665         {
6666             \cs_if_exist_p:c
6667             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6668         }
6669         { \bool_set_true:N \l_tmpa_bool }
6670         {
6671             \bool_if:NF \l_tmpa_bool
6672             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6673         }
6674     }

```

Now, we loop over the rows.

```

6675     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6676     {

```

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

```

6677         \bool_set_false:N \l_tmpa_bool
6678         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6679         {
6680             \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6681             \bool_lazy_or:nnTF
6682             \l_tmpb_bool
6683             {
6684                 \cs_if_exist_p:c
6685                 { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6686             }
6687             { \bool_set_true:N \l_tmpa_bool }
6688             {
6689                 \bool_if:NF \l_tmpa_bool
6690                 {
6691                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6692                     \seq_put_right:Nn
6693                     \l_@@_corners_cells_seq
6694                     { ##1 - #####1 }
6695                 }
6696             }
6697         }
6698     }
6699 }

```

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`).

```

6700 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6701 {
6702   \int_set:Nn \l_tmpa_int { #1 }
6703   \int_set:Nn \l_tmpb_int { #2 }
6704   \bool_set_false:N \l_tmpb_bool
6705   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6706     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6707 }
6708 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6709 {
6710   \int_compare:nNnF { #3 } > { #1 }
6711   {
6712     \int_compare:nNnF { #1 } > { #5 }
6713     {
6714       \int_compare:nNnF { #4 } > { #2 }
6715       {
6716         \int_compare:nNnF { #2 } > { #6 }
6717         { \bool_set_true:N \l_tmpb_bool }
6718       }
6719     }
6720   }
6721 }

```

## 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.

```

6722 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6723 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6724 {
6725   auto-columns-width .code:n =
6726   {
6727     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6728     \dim_gzero_new:N \g_@@_max_cell_width_dim
6729     \bool_set_true:N \l_@@_auto_columns_width_bool
6730   }
6731 }
6732 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6733 {
6734   \int_gincr:N \g_@@_NiceMatrixBlock_int
6735   \dim_zero:N \l_@@_columns_width_dim
6736   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6737   \bool_if:NT \l_@@_block_auto_columns_width_bool
6738   {
6739     \cs_if_exist:cT
6740     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6741     {
6742       % is \exp_args:NNe mandatory?
6743       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6744       {
6745         \use:c
6746         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6747       }
6748     }

```

```

6749     }
6750 }

```

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`).

```

6751 {
6752   \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.

```

6753   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6754   {
6755     \bool_if:NT \l_@@_block_auto_columns_width_bool
6756     {
6757       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6758       \iow_shipout:Nx \@mainaux
6759       {
6760         \cs_gset:cpn
6761         { @@ _ 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.

```

6762         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6763       }
6764       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6765     }
6766   }
6767   \ignorespacesafterend
6768 }

```

## 26 The extra nodes

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

```

6769 \cs_generate_variant:Nn \dim_min:nn { v n }
6770 \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).

```

6771 \cs_new_protected:Npn \@@_create_extra_nodes:
6772 {
6773   \bool_if:nTF \l_@@_medium_nodes_bool
6774   {
6775     \bool_if:NTF \l_@@_large_nodes_bool
6776     \@@_create_medium_and_large_nodes:
6777     \@@_create_medium_nodes:
6778   }
6779   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6780 }

```

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  $\backslash c\_max\_dim$  or  $-\backslash c\_max\_dim$ .

```

6781 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6782 {
6783   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6784   {
6785     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6786     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6787     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6788     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6789   }
6790   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6791   {
6792     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6793     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6794     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6795     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6796   }

```

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

```

6797   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6798   {
6799     \int_step_variable:nnNn
6800     \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.

```

6801     {
6802       \cs_if_exist:cT
6803       { 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  $\backslash pgf@x$  and  $\backslash pgf@y$ .

```

6804     {
6805       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6806       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6807       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6808       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6809       {
6810         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6811         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6812       }

```

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

```

6813       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6814       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6815       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6816       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6817       {
6818         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6819         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6820       }
6821     }
6822   }
6823 }

```

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

```

6824 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6825 {
6826   \dim_compare:nnNt
6827   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6828   {
6829     \@@_qpoint:n { row - \@@_i: - base }
6830     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6831     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6832   }
6833 }
6834 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6835 {
6836   \dim_compare:nnNt
6837   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6838   {
6839     \@@_qpoint:n { col - \@@_j: }
6840     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6841     \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6842   }
6843 }
6844 }

```

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

```

6845 \cs_new_protected:Npn \@@_create_medium_nodes:
6846 {
6847   \pgfpicture
6848   \pgfrememberpicturepositiononpagetrue
6849   \pgf@relevantforpicturesizefalse
6850   \@@_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”.

```

6851   \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6852   \@@_create_nodes:
6853   \endpgfpicture
6854 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones<sup>14</sup>. 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:`.

```

6855 \cs_new_protected:Npn \@@_create_large_nodes:
6856 {
6857   \pgfpicture
6858   \pgfrememberpicturepositiononpagetrue
6859   \pgf@relevantforpicturesizefalse
6860   \@@_computations_for_medium_nodes:
6861   \@@_computations_for_large_nodes:
6862   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6863   \@@_create_nodes:
6864   \endpgfpicture
6865 }
6866 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6867 {

```

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6868 \pgfpicture
6869 \pgfrememberpicturepositiononpagetrue
6870 \pgf@relevantforpicturesizefalse
6871 \@@_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”.

```

6872 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6873 \@@_create_nodes:
6874 \@@_computations_for_large_nodes:
6875 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6876 \@@_create_nodes:
6877 \endpgfpicture
6878 }

```

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.

```

6879 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6880 {
6881 \int_set_eq:NN \l_@@_first_row_int \c_one_int
6882 \int_set_eq:NN \l_@@_first_col_int \c_one_int

```

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`.

```

6883 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6884 {
6885 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6886 {
6887 (
6888 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6889 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6890 )
6891 / 2
6892 }
6893 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6894 { l_@@_row _ \@@_i: _ min _ dim }
6895 }
6896 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6897 {
6898 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6899 {
6900 (
6901 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6902 \dim_use:c
6903 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6904 )
6905 / 2
6906 }
6907 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6908 { l_@@_column _ \@@_j: _ max _ dim }
6909 }

```

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

```

6910 \dim_sub:cn
6911 { l_@@_column _ 1 _ min _ dim }
6912 \l_@@_left_margin_dim
6913 \dim_add:cn
6914 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6915 \l_@@_right_margin_dim
6916 }

```

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).

```

6917 \cs_new_protected:Npn \@@_create_nodes:
6918 {
6919   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6920   {
6921     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6922     {

```

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

```

6923       \@@_pgf_rect_node:nnnnn
6924       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6925       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6926       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6927       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6928       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6929       \str_if_empty:NF \l_@@_name_str
6930       {
6931         \pgfnodealias
6932         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6933         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6934       }
6935     }
6936   }

```

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  $n$ .

```

6937   \seq_map_pairwise_function:NNN
6938   \g_@@_multicolumn_cells_seq
6939   \g_@@_multicolumn_sizes_seq
6940   \@@_node_for_multicolumn:nn
6941 }

```

```

6942 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6943 {
6944   \cs_set_nopar:Npn \@@_i: { #1 }
6945   \cs_set_nopar:Npn \@@_j: { #2 }
6946 }

```

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”).

```

6947 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6948 {
6949   \@@_extract_coords_values: #1 \q_stop
6950   \@@_pgf_rect_node:nnnnn
6951   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6952   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6953   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6954   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: + #2 - 1 } _max_dim } }
6955   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6956   \str_if_empty:NF \l_@@_name_str
6957   {
6958     \pgfnodealias
6959     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6960     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6961   }
6962 }

```

## 27 The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

```

6963 \keys_define:n { NiceMatrix / Block / FirstPass }
6964 {
6965   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6966   l .value_forbidden:n = true ,
6967   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6968   r .value_forbidden:n = true ,
6969   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6970   c .value_forbidden:n = true ,
6971   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6972   L .value_forbidden:n = true ,
6973   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6974   R .value_forbidden:n = true ,
6975   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6976   C .value_forbidden:n = true ,
6977   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6978   t .value_forbidden:n = true ,
6979   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6980   T .value_forbidden:n = true ,
6981   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6982   b .value_forbidden:n = true ,
6983   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
6984   B .value_forbidden:n = true ,
6985   color .code:n =
6986     \@@_color:n { #1 }
6987     \tl_set_rescan:Nnn
6988       \l_@@_draw_tl
6989       { \char_set_catcode_other:N ! }
6990       { #1 } ,
6991   color .value_required:n = true ,
6992   respect-arraystretch .code:n =
6993     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
6994   respect-arraystretch .value_forbidden:n = true ,
6995 }

```

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.

```

6996 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6997 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6998 {

```

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).

```

6999   \peek_remove_spaces:n
7000   {
7001     \tl_if_blank:nTF { #2 }
7002     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7003     {
7004       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7005       \@@_Block_i_czech \@@_Block_i
7006       #2 \q_stop
7007     }
7008     { #1 } { #3 } { #4 }

```

```

7009     }
7010 }

```

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

```

7011 \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`.

```

7012 {
7013   \char_set_catcode_active:N -
7014   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7015 }

```

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.

```

7016 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7017 {

```

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).

```

7018   \bool_lazy_or:nnTF
7019     { \tl_if_blank_p:n { #1 } }
7020     { \str_if_eq_p:nn { #1 } { * } }
7021     { \int_set:Nn \l_tmpa_int { 100 } }
7022     { \int_set:Nn \l_tmpa_int { #1 } }
7023   \bool_lazy_or:nnTF
7024     { \tl_if_blank_p:n { #2 } }
7025     { \str_if_eq_p:nn { #2 } { * } }
7026     { \int_set:Nn \l_tmpb_int { 100 } }
7027     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7028   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7029     {
7030       \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7031         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7032         { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7033     }
7034     { \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.

```

7035   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
7036   \tl_set:Nx \l_tmpa_tl
7037     {
7038     { \int_use:N \c@iRow }
7039     { \int_use:N \c@jCol }
7040     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7041     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7042   }

```

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).

```

7043   \bool_if:nTF
7044   {
7045     (
7046       \int_compare_p:nNn \l_tmpa_int = \c_one_int
7047       ||
7048       \int_compare_p:nNn \l_tmpb_int = \c_one_int
7049     )
7050     && ! \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.

```

7051     && ! \l_@@_X_bool
7052   }
7053   { \exp_args:Nee \@@_Block_iv:nnnnn }
7054   { \exp_args:Nee \@@_Block_v:nnnnn }
7055   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7056 }

```

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.

```

7057 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7058 {
7059   \int_gincr:N \g_@@_block_box_int
7060   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7061   {
7062     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7063     {
7064       \@@_actually_diagbox:nnnnnn
7065       { \int_use:N \c_iRow }
7066       { \int_use:N \c_jCol }
7067       { \int_eval:n { \c_iRow + #1 - 1 } }
7068       { \int_eval:n { \c_jCol + #2 - 1 } }
7069       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7070       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7071     }
7072   }
7073   \box_gclear_new:c
7074   { 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!

```

7075   \hbox_gset:cn
7076   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7077   {

```

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}`).

```

7078 \tl_if_empty:NTF \l_@@_color_tl
7079 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7080 { \@@_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`.

```

7081 \int_compare:nNnT { #1 } = \c_one_int
7082 {
7083   \int_if_zero:nTF \c@iRow
7084   \l_@@_code_for_first_row_tl
7085   {
7086     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7087     \l_@@_code_for_last_row_tl
7088   }
7089   \g_@@_row_style_tl
7090 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7091 \@@_reset_arraystretch:
7092 \dim_zero:N \extrarowheight

```

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

```

7093 #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.).

```

7094 \@@_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}`.

```

7095 \bool_if:NTF \l_@@_tabular_bool
7096 {
7097   \bool_lazy_all:nTF
7098   {
7099     { \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.

```

7100 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7101 { ! \g_@@_rotate_bool }
7102 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7103 {
7104   \use:e
7105   {
7106     \exp_not:N \begin { minipage }%
7107     [ \str_lowercase:V \l_@@_vpos_block_str ]
7108     { \l_@@_col_width_dim }
7109     \str_case:on \l_@@_hpos_block_str
7110     { c \centering r \raggedleft l \raggedright }
7111   }
7112   #5
7113   \end { minipage }
7114 }

```

In the other cases, we use a `{tabular}`.

```

7115         {
7116             \use:e
7117             {
7118                 \exp_not:N \begin { tabular }%
7119                 [ \str_lowercase:V \l_@@_vpos_block_str ]
7120                 { @ { } \l_@@_hpos_block_str @ { } }
7121             }
7122             #5
7123             \end { tabular }
7124         }
7125     }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7126     {
7127         \c_math_toggle_token
7128         \use:e
7129         {
7130             \exp_not:N \begin { array }%
7131             [ \str_lowercase:V \l_@@_vpos_block_str ]
7132             { @ { } \l_@@_hpos_block_str @ { } }
7133         }
7134         #5
7135         \end { array }
7136         \c_math_toggle_token
7137     }
7138 }

```

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).

```

7139     \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.

```

7140     \int_compare:nNnT { #2 } = \c_one_int
7141     {
7142         \dim_gset:Nn \g_@@_blocks_wd_dim
7143         {
7144             \dim_max:nn
7145             \g_@@_blocks_wd_dim
7146             {
7147                 \box_wd:c
7148                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7149             }
7150         }
7151     }

```

If we are in a mono-row block and if that block has no vertical option for the position<sup>15</sup>, we take into account the height and the depth of that block for the height and the depth of the row.

```

7152     \str_if_eq:VnT \l_@@_vpos_block_str { c }
7153     {
7154         \int_compare:nNnT { #1 } = \c_one_int
7155         {
7156             \dim_gset:Nn \g_@@_blocks_ht_dim
7157             {
7158                 \dim_max:nn
7159                 \g_@@_blocks_ht_dim

```

---

<sup>15</sup>If 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`.

```

7160         {
7161             \box_ht:c
7162             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7163         }
7164     }
7165     \dim_gset:Nn \g_@@_blocks_dp_dim
7166     {
7167         \dim_max:nn
7168         \g_@@_blocks_dp_dim
7169         {
7170             \box_dp:c
7171             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7172         }
7173     }
7174 }
7175 }
7176 \seq_gput_right:Nx \g_@@_blocks_seq
7177 {
7178     \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.

```

7179     {
7180         \exp_not:n { #3 } ,
7181         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7182         \bool_if:NT \g_@@_rotate_bool
7183         {
7184             \bool_if:NTF \g_@@_rotate_c_bool
7185             { v-center }
7186             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7187         }
7188     }
7189     {
7190         \box_use_drop:c
7191         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7192     }
7193 }
7194 }
7195 \bool_set_false:N \g_@@_rotate_c_bool
7196 }

7197 \cs_new:Npn \@@_adjust_hpos_rotate:
7198 {
7199     \bool_if:NT \g_@@_rotate_bool
7200     {
7201         \str_set:Nx \l_@@_hpos_block_str
7202         {
7203             \bool_if:NTF \g_@@_rotate_c_bool
7204             { c }
7205             {
7206                 \str_case:onF \l_@@_vpos_block_str
7207                 { b l B l t r T r }
7208                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7209             }
7210         }
7211     }
7212 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7213 \cs_new_protected:Npn \@@_rotate_box_of_block:
7214 {
7215   \box_grotate:cn
7216   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7217   { 90 }
7218   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7219   {
7220     \vbox_gset_top:cn
7221     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7222     {
7223       \skip_vertical:n { 0.8 ex }
7224       \box_use:c
7225       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7226     }
7227   }
7228   \bool_if:NT \g_@@_rotate_c_bool
7229   {
7230     \hbox_gset:cn
7231     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7232     {
7233       \c_math_toggle_token
7234       \vcenter
7235       {
7236         \box_use:c
7237         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7238       }
7239       \c_math_toggle_token
7240     }
7241   }
7242 }

```

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.

```

7243 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7244 {
7245   \seq_gput_right:Nx \g_@@_blocks_seq
7246   {
7247     \l_tmpa_tl
7248     { \exp_not:n { #3 } }
7249     {
7250       \bool_if:NTF \l_@@_tabular_bool
7251       {
7252         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7253   \@@_reset_arraystretch:
7254   \exp_not:n
7255   {
7256     \dim_zero:N \extrarowheight
7257     #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).

```

7258         \use:e
7259         {
7260             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7261             { @ { } \l_@@_hpos_block_str @ { } }
7262         }
7263         #5
7264         \end { tabular }
7265     }
7266     \group_end:
7267 }

```

When we are *not* in an environments {NiceTabular} (or similar).

```

7268     {
7269         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7270         \@@_reset_arraystretch:
7271         \exp_not:n
7272         {
7273             \dim_zero:N \extrarowheight
7274             #4
7275             \c_math_toggle_token
7276             \use:e
7277             {
7278                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7279                 { @ { } \l_@@_hpos_block_str @ { } }
7280             }
7281             #5
7282             \end { array }
7283             \c_math_toggle_token
7284         }
7285         \group_end:
7286     }
7287 }
7288 }
7289 }

```

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).

```

7290 \keys_define:nn { NiceMatrix / Block / SecondPass }
7291 {
7292     tikz .code:n =
7293         \IfPackageLoadedTF { tikz }
7294         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7295         { \@@_error:n { tikz~key~without~tikz } } ,
7296     tikz .value_required:n = true ,
7297     fill .code:n =
7298         \tl_set_rescan:Nnn
7299         \l_@@_fill_tl
7300         { \char_set_catcode_other:N ! }
7301         { #1 } ,
7302     fill .value_required:n = true ,
7303     opacity .tl_set:N = \l_@@_opacity_tl ,
7304     opacity .value_required:n = true ,
7305     draw .code:n =
7306         \tl_set_rescan:Nnn
7307         \l_@@_draw_tl
7308         { \char_set_catcode_other:N ! }
7309         { #1 } ,
7310     draw .default:n = default ,
7311     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7312     rounded-corners .default:n = 4 pt ,

```

```

7313 color .code:n =
7314   \@@_color:n { #1 }
7315   \tl_set_rescan:Nnn
7316   \l_@@_draw_tl
7317   { \char_set_catcode_other:N ! }
7318   { #1 } ,
7319 borders .clist_set:N = \l_@@_borders_clist ,
7320 borders .value_required:n = true ,
7321 hvlines .meta:n = { vl_lines , hl_lines } ,
7322 vl_lines .bool_set:N = \l_@@_vl_lines_block_bool ,
7323 vl_lines .default:n = true ,
7324 hl_lines .bool_set:N = \l_@@_hl_lines_block_bool ,
7325 hl_lines .default:n = true ,
7326 line-width .dim_set:N = \l_@@_line_width_dim ,
7327 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7328 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7329 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7330 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7331 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7332   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7333 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7334   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7335 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7336   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7337 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7338 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7339 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7340 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7341 v-center .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7342 v-center .value_forbidden:n = true ,
7343 name .tl_set:N = \l_@@_block_name_str ,
7344 name .value_required:n = true ,
7345 name .initial:n = ,
7346 respect-arraystretch .code:n =
7347   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7348 respect-arraystretch .value_forbidden:n = true ,
7349 transparent .bool_set:N = \l_@@_transparent_bool ,
7350 transparent .default:n = true ,
7351 transparent .initial:n = false ,
7352 unknown .code:n = \@@_error:n { Unknown-key-for-Block }
7353 }

```

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.

```

7354 \cs_new_protected:Npn \@@_draw_blocks:
7355 {
7356   \cs_set_eq:NN \ialign \@@_old_ialign:
7357   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
7358 }
7359 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7360 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7361 \int_zero_new:N \l_@@_last_row_int
7362 \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.

```

7363 \int_compare:nNnTF { #3 } > { 99 }
7364   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7365   { \int_set:Nn \l_@@_last_row_int { #3 } }
7366 \int_compare:nNnTF { #4 } > { 99 }
7367   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7368   { \int_set:Nn \l_@@_last_col_int { #4 } }
7369 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7370   {
7371     \bool_lazy_and:nnTF
7372       \l_@@_preamble_bool
7373       {
7374         \int_compare_p:n
7375           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7376       }
7377       {
7378         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7379         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7380         \@@_msg_redirect_name:nn { columns-not-used } { none }
7381       }
7382       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7383   }
7384   {
7385     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7386       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7387       { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7388   }
7389 }

```

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

```

7390 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7391   {

```

The group is for the keys.

```

7392   \group_begin:
7393   \int_compare:nNnT { #1 } = { #3 }
7394     { \str_set:Nn \l_@@_vpos_block_str { t } }
7395   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7396   \bool_if:NT \l_@@_vlines_block_bool
7397     {
7398       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7399       {
7400         \@@_vlines_block:nnn
7401           { \exp_not:n { #5 } }
7402           { #1 - #2 }
7403           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7404       }
7405     }
7406   \bool_if:NT \l_@@_hlines_block_bool
7407     {
7408       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7409       {
7410         \@@_hlines_block:nnn
7411           { \exp_not:n { #5 } }
7412           { #1 - #2 }
7413           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7414       }
7415     }
7416   \bool_if:NF \l_@@_transparent_bool

```

```

7417 {
7418   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7419   {

```

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).

```

7420     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7421     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7422   }
7423 }

```

```

7424 \tl_if_empty:NF \l_@@_draw_tl
7425 {
7426   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7427   { \@@_error:n { hlines-with-color } }
7428 }

```

```

7429 \tl_if_empty:NF \l_@@_draw_tl
7430 {
7431   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7432   {
7433     \@@_stroke_block:nnn
7434     { \exp_not:n { #5 } } % #5 are the options
7435     { #1 - #2 }
7436     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7437   }
7438   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7439   { { #1 } { #2 } { #3 } { #4 } }
7440 }

```

```

7441 \clist_if_empty:NF \l_@@_borders_clist
7442 {
7443   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7444   {
7445     \@@_stroke_borders_block:nnn
7446     { \exp_not:n { #5 } }
7447     { #1 - #2 }
7448     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7449   }
7450 }

```

```

7451 \tl_if_empty:NF \l_@@_fill_tl
7452 {
7453   \tl_if_empty:NF \l_@@_opacity_tl
7454   {
7455     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7456       {
7457         \tl_set:Nx \l_@@_fill_tl
7458         {
7459           [ opacity = \l_@@_opacity_tl ,
7460           \tl_tail:o \l_@@_fill_tl
7461         ]
7462       }
7463       {
7464         \tl_set:Nx \l_@@_fill_tl
7465         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7466       }
7467     ]
7468     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7469     {
7470       \exp_not:N \roundedrectanglecolor
7471       \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7472         { \l_@@_fill_tl }

```

```

7473         { { \l_@@_fill_tl } }
7474     { #1 - #2 }
7475     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7476     { \dim_use:N \l_@@_rounded_corners_dim }
7477 }
7478 }
7479 \seq_if_empty:NF \l_@@_tikz_seq
7480 {
7481     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7482     {
7483         \@@_block_tikz:nnnnn
7484         { #1 }
7485         { #2 }
7486         { \int_use:N \l_@@_last_row_int }
7487         { \int_use:N \l_@@_last_col_int }
7488         { \seq_use:Nn \l_@@_tikz_seq { , } }
7489     }
7490 }

7491 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7492 {
7493     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7494     {
7495         \@@_actually_diagbox:nnnnnn
7496         { #1 }
7497         { #2 }
7498         { \int_use:N \l_@@_last_row_int }
7499         { \int_use:N \l_@@_last_col_int }
7500         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7501     }
7502 }

7503 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7504 \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.

```

7505 \pgfpicture
7506 \pgfrememberpicturepositiononpagetrue
7507 \pgf@relevantforpicturesizefalse
7508 \@@_qpoint:n { row - #1 }
7509 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7510 \@@_qpoint:n { col - #2 }

```

```

7511 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7512 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7513 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7514 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7515 \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.

```

7516 \@@_pgf_rect_node:nnnnn
7517 { \@@_env: - #1 - #2 - block }
7518 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7519 \str_if_empty:NF \l_@@_block_name_str
7520 {
7521   \pgfnodealias
7522   { \@@_env: - \l_@@_block_name_str }
7523   { \@@_env: - #1 - #2 - block }
7524   \str_if_empty:NF \l_@@_name_str
7525   {
7526     \pgfnodealias
7527     { \l_@@_name_str - \l_@@_block_name_str }
7528     { \@@_env: - #1 - #2 - block }
7529   }
7530 }

```

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.

```

7531 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7532 {
7533   \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.

```

7534 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7535 {

```

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.

```

7536 \cs_if_exist:cT
7537 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7538 {
7539   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7540   {
7541     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7542     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7543   }
7544 }
7545 }

```

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.

```

7546 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7547 {
7548   \@@_qpoint:n { col - #2 }
7549   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7550 }
7551 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7552 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7553 {
7554   \cs_if_exist:cT
7555   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7556   {

```

```

7557         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7558         {
7559             \pgfpointanchor
7560             { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7561             { east }
7562             \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7563         }
7564     }
7565 }
7566 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7567 {
7568     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7569     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7570 }
7571 \@@_pgf_rect_node:nnnnn
7572 { \@@_env: - #1 - #2 - block - short }
7573 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7574 }

```

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.

```

7575 \bool_if:NT \l_@@_medium_nodes_bool
7576 {
7577     \@@_pgf_rect_node:nnn
7578     { \@@_env: - #1 - #2 - block - medium }
7579     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7580     {
7581         \pgfpointanchor
7582         { \@@_env:
7583             - \int_use:N \l_@@_last_row_int
7584             - \int_use:N \l_@@_last_col_int - medium
7585         }
7586         { south-east }
7587     }
7588 }

```

Now, we will put the label of the block.

```

7589 \bool_lazy_any:nTF
7590 {
7591     { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7592     { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7593     { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7594 }
7595 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7596     \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`.

```

7597     \bool_if:nT \g_@@_last_col_found_bool
7598     {
7599         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7600         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7601     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7602     \tl_set:Nx \l_tmpa_tl
7603     {
7604         \str_case:on \l_@@_vpos_block_str
7605         {
7606             c {
7607                 \str_case:on \l_@@_hpos_block_str
7608                 {

```

```

7609             c { center }
7610             l { west }
7611             r { east }
7612         }
7613     }
7614 }
7615 T {
7616     \str_case:on \l_@@_hpos_block_str
7617     {
7618         c { north }
7619         l { north~west }
7620         r { north~east }
7621     }
7622 }
7623 }
7624 B {
7625     \str_case:on \l_@@_hpos_block_str
7626     {
7627         c { south}
7628         l { south~west }
7629         r { south~east }
7630     }
7631 }
7632 }
7633 }
7634 }
7635 \pgftransformshift
7636 {
7637     \pgfpointanchor
7638     {
7639         \@@_env: - #1 - #2 - block
7640         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7641     }
7642     { \l_tmpa_tl }
7643 }
7644 \pgfset
7645 {
7646     inner~xsep = \c_zero_dim ,
7647     inner~ysep = \c_zero_dim
7648 }
7649 \pgfnode
7650 { rectangle }
7651 { \l_tmpa_tl }
7652 { \box_use_drop:N \l_@@_cell_box } { } { }
7653 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7654 {
7655     \pgfextracty \l_tmpa_dim
7656     {
7657         \@@_qpoint:n
7658         {
7659             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7660             - base
7661         }
7662     }
7663     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth } % added 2023-02-21

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

7664 \pgfpointanchor
7665 {
7666     \@@_env: - #1 - #2 - block
7667     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }

```

```

7668     }
7669     {
7670         \str_case:on \l_@@_hpos_block_str
7671         {
7672             c { center }
7673             l { west }
7674             r { east }
7675         }
7676     }

```

We put the label of the block which has been composed in \l\_@@\_cell\_box.

```

7677     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7678     \pgfset { inner~sep = \c_zero_dim }
7679     \pgfnode
7680     { rectangle }
7681     {
7682         \str_case:on \l_@@_hpos_block_str
7683         {
7684             c { base }
7685             l { base~west }
7686             r { base~east }
7687         }
7688     }
7689     { \box_use_drop:N \l_@@_cell_box } { } { }
7690 }
7691 \endpgfpicture
7692 \group_end:
7693 }

```

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).

```

7694 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7695 {
7696     \group_begin:
7697     \tl_clear:N \l_@@_draw_tl
7698     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7699     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7700     \pgfpicture
7701     \pgfrememberpicturerepositiononpagetrue
7702     \pgf@relevantforpicturesizefalse
7703     \tl_if_empty:NF \l_@@_draw_tl
7704     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7705         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7706         { \CT@arc@ }
7707         { \@@_color:o \l_@@_draw_tl }
7708     }
7709     \pgfsetcornersarced
7710     {
7711         \pgfpoint
7712         { \l_@@_rounded_corners_dim }
7713         { \l_@@_rounded_corners_dim }
7714     }
7715     \@@_cut_on_hyphen:w #2 \q_stop
7716     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7717     {
7718         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7719         {
7720             \@@_qpoint:n { row - \l_tmpa_tl }
7721             \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

7722 \@@_qpoint:n { col - \l_tmpb_tl }
7723 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7724 \@@_cut_on_hyphen:w #3 \q_stop
7725 \int_compare:nNnT \l_tmpa_tl > \c@iRow
7726 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7727 \int_compare:nNnT \l_tmpb_tl > \c@jCol
7728 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7729 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7730 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7731 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7732 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7733 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7734 \pgfpathrectanglecorners
7735 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7736 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7737 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7738 { \pgfusepathqstroke }
7739 { \pgfusepath { stroke } }
7740 }
7741 }
7742 \endpgfpicture
7743 \group_end:
7744 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7745 \keys_define:nn { NiceMatrix / BlockStroke }
7746 {
7747   color .tl_set:N = \l_@@_draw_tl ,
7748   draw .code:n =
7749     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7750   draw .default:n = default ,
7751   line-width .dim_set:N = \l_@@_line_width_dim ,
7752   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7753   rounded-corners .default:n = 4 pt
7754 }

```

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).

```

7755 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7756 {
7757   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7758   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7759   \@@_cut_on_hyphen:w #2 \q_stop
7760   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7761   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7762   \@@_cut_on_hyphen:w #3 \q_stop
7763   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7764   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7765   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7766   {
7767     \use:e
7768     {
7769       \@@_vline:n
7770       {
7771         position = ##1 ,
7772         start = \l_@@_tmpc_tl ,
7773         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7774         total-width = \dim_use:N \l_@@_line_width_dim
7775       }
7776     }
7777   }
7778 }

```

```

7779 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7780 {
7781   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7782   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7783   \@@_cut_on_hyphen:w #2 \q_stop
7784   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7785   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7786   \@@_cut_on_hyphen:w #3 \q_stop
7787   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7788   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7789   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7790   {
7791     \use:e
7792     {
7793       \@@_hline:n
7794       {
7795         position = ##1 ,
7796         start = \l_@@_tmpd_tl ,
7797         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7798         total-width = \dim_use:N \l_@@_line_width_dim
7799       }
7800     }
7801   }
7802 }

```

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).

```

7803 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7804 {
7805   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7806   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7807   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7808   { \@@_error:n { borders~forbidden } }
7809   {
7810     \tl_clear_new:N \l_@@_borders_tikz_tl
7811     \keys_set:nV
7812       { NiceMatrix / OnlyForTikzInBorders }
7813     \l_@@_borders_clist
7814     \@@_cut_on_hyphen:w #2 \q_stop
7815     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7816     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7817     \@@_cut_on_hyphen:w #3 \q_stop
7818     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7819     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7820     \@@_stroke_borders_block_i:
7821   }
7822 }
7823 \hook_gput_code:nnn { begindocument } { . }
7824 {
7825   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7826   {
7827     \c_@@_pgfortikzpicture_tl
7828     \@@_stroke_borders_block_ii:
7829     \c_@@_endpgfortikzpicture_tl
7830   }
7831 }
7832 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7833 {
7834   \pgfrememberpicturepositiononpagetrue
7835   \pgf@relevantforpicturesizefalse
7836   \CT@arc@

```

```

7837 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7838 \clist_if_in:NnT \l_@@_borders_clist { right }
7839   { \@@_stroke_vertical:n \l_tmpb_tl }
7840 \clist_if_in:NnT \l_@@_borders_clist { left }
7841   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7842 \clist_if_in:NnT \l_@@_borders_clist { bottom }
7843   { \@@_stroke_horizontal:n \l_tmpa_tl }
7844 \clist_if_in:NnT \l_@@_borders_clist { top }
7845   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7846 }
7847 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7848 {
7849   tikz .code:n =
7850     \cs_if_exist:NTF \tikzpicture
7851       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7852       { \@@_error:n { tikz~in~borders~without~tikz } } ,
7853   tikz .value_required:n = true ,
7854   top .code:n = ,
7855   bottom .code:n = ,
7856   left .code:n = ,
7857   right .code:n = ,
7858   unknown .code:n = \@@_error:n { bad~border }
7859 }

```

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).

```

7860 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7861 {
7862   \@@_qpoint:n \l_@@_tmpc_tl
7863   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7864   \@@_qpoint:n \l_tmpa_tl
7865   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7866   \@@_qpoint:n { #1 }
7867   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7868   {
7869     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7870     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7871     \pgfusepathqstroke
7872   }
7873   {
7874     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7875     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7876   }
7877 }

```

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).

```

7878 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7879 {
7880   \@@_qpoint:n \l_@@_tmpd_tl
7881   \clist_if_in:NnTF \l_@@_borders_clist { left }
7882     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7883     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7884   \@@_qpoint:n \l_tmpb_tl
7885   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7886   \@@_qpoint:n { #1 }
7887   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7888   {
7889     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7890     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7891     \pgfusepathqstroke
7892   }
7893   {

```

```

7894     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7895     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7896 }
7897 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7898 \keys_define:nn { NiceMatrix / BlockBorders }
7899 {
7900   borders .clist_set:N = \l_@@_borders_clist ,
7901   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7902   rounded-corners .default:n = 4 pt ,
7903   line-width .dim_set:N = \l_@@_line_width_dim
7904 }

```

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.

```

7905 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7906 {
7907   \begin { tikzpicture }
7908   \@@_clip_with_rounded_corners:
7909   \clist_map_inline:nn { #5 }
7910   {
7911     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7912     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7913     (
7914       [
7915         xshift = \dim_use:N \l_@@_offset_dim ,
7916         yshift = - \dim_use:N \l_@@_offset_dim
7917       ]
7918       #1 -| #2
7919     )
7920     rectangle
7921     (
7922       [
7923         xshift = - \dim_use:N \l_@@_offset_dim ,
7924         yshift = \dim_use:N \l_@@_offset_dim
7925       ]
7926       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7927     ) ;
7928   }
7929   \end { tikzpicture }
7930 }
7931 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7932 \keys_define:nn { NiceMatrix / SpecialOffset }
7933 { offset .dim_set:N = \l_@@_offset_dim }

```

## 28 How to draw the dotted lines transparently

```

7934 \cs_set_protected:Npn \@@_renew_matrix:
7935 {
7936   \RenewDocumentEnvironment { pmatrix } { }
7937   { \pNiceMatrix }
7938   { \endpNiceMatrix }
7939   \RenewDocumentEnvironment { vmatrix } { }
7940   { \vNiceMatrix }

```

```

7941     { \endvNiceMatrix }
7942 \RenewDocumentEnvironment { Vmatrix } { }
7943     { \VNiceMatrix }
7944     { \endVNiceMatrix }
7945 \RenewDocumentEnvironment { bmatrix } { }
7946     { \bNiceMatrix }
7947     { \endbNiceMatrix }
7948 \RenewDocumentEnvironment { Bmatrix } { }
7949     { \BNiceMatrix }
7950     { \endBNiceMatrix }
7951 }

```

## 29 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```

7952 \keys_define:nn { NiceMatrix / Auto }
7953 {
7954     columns-type .tl_set:N = \l_@@_columns_type_tl ,
7955     columns-type .value_required:n = true ,
7956     l .meta:n = { columns-type = l } ,
7957     r .meta:n = { columns-type = r } ,
7958     c .meta:n = { columns-type = c } ,
7959     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7960     delimiters / color .value_required:n = true ,
7961     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7962     delimiters / max-width .default:n = true ,
7963     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7964     delimiters .value_required:n = true ,
7965     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7966     rounded-corners .default:n = 4 pt
7967 }
7968 \NewDocumentCommand \AutoNiceMatrixWithDelims
7969 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7970 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7971 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7972 {

```

The group is for the protection of the keys.

```

7973 \group_begin:
7974 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
7975 \use:e
7976 {
7977     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7978     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
7979     [ \exp_not:o \l_tmpa_tl ]
7980 }
7981 \int_if_zero:nT \l_@@_first_row_int
7982 {
7983     \int_if_zero:nT \l_@@_first_col_int { & }
7984     \prg_replicate:nn { #4 - 1 } { & }
7985     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7986 }
7987 \prg_replicate:nn { #3 }
7988 {
7989     \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).

```

7990     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7991     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\

```

```

7992     }
7993     \int_compare:nNtT \l_@@_last_row_int > { -2 }
7994     {
7995         \int_if_zero:nT \l_@@_first_col_int { & }
7996         \prg_replicate:nn { #4 - 1 } { & }
7997         \int_compare:nNtT \l_@@_last_col_int > { -1 } { & } \\
7998     }
7999     \end { NiceArrayWithDelims }
8000     \group_end:
8001 }

8002 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8003 {
8004     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8005     {
8006         \bool_gset_true:N \g_@@_delims_bool
8007         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8008         \AutoNiceMatrixWithDelims { #2 } { #3 }
8009     }
8010 }

8011 \@@_define_com:nnn p ( )
8012 \@@_define_com:nnn b [ ]
8013 \@@_define_com:nnn v | |
8014 \@@_define_com:nnn V \l \l
8015 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8016 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8017 {
8018     \group_begin:
8019     \bool_gset_false:N \g_@@_delims_bool
8020     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8021     \group_end:
8022 }

```

## 30 The redefinition of the command `\dotfill`

```

8023 \cs_set_eq:NN \@@_old_dotfill \dotfill
8024 \cs_new_protected:Npn \@@_dotfill:
8025 {

```

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}`).

```

8026     \@@_old_dotfill
8027     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8028 }

```

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`.

```

8029 \cs_new_protected:Npn \@@_dotfill_i:
8030 { \dim_compare:nNtT { \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.

```

8031 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8032 {
8033   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8034   {
8035     \@@_actually_diagbox:nnnnnn
8036     { \int_use:N \c@iRow }
8037     { \int_use:N \c@jCol }
8038     { \int_use:N \c@iRow }
8039     { \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.

```

8040     { \g_@@_row_style_tl \exp_not:n { #1 } }
8041     { \g_@@_row_style_tl \exp_not:n { #2 } }
8042   }

```

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`.

```

8043   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8044   {
8045     { \int_use:N \c@iRow }
8046     { \int_use:N \c@jCol }
8047     { \int_use:N \c@iRow }
8048     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8049     { }
8050   }
8051 }

```

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.

```

8052 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8053 {
8054   \pgfpicture
8055   \pgf@relevantforpicturesizefalse
8056   \pgfrememberpicturepositiononpagetrue
8057   \@@_qpoint:n { row - #1 }
8058   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8059   \@@_qpoint:n { col - #2 }
8060   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8061   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8062   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8063   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8064   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8065   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8066   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8067   {

```

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.

```

8068   \CT@arc@
8069   \pgfsetroundcap
8070   \pgfusepathqstroke
8071 }
8072 \pgfset { inner~sep = 1 pt }
8073 \pgfscope
8074 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }

```

```

8075 \pgfnode { rectangle } { south-west }
8076 {
8077   \begin { minipage } { 20 cm }
8078   \@@_math_toggle: #5 \@@_math_toggle:
8079   \end { minipage }
8080 }
8081 { }
8082 { }
8083 \endpgfscope
8084 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8085 \pgfnode { rectangle } { north-east }
8086 {
8087   \begin { minipage } { 20 cm }
8088   \raggedleft
8089   \@@_math_toggle: #6 \@@_math_toggle:
8090   \end { minipage }
8091 }
8092 { }
8093 { }
8094 \endpgfpicture
8095 }

```

## 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`.

```

8096 \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 `\`.

```

8097 \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`.

```

8098 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8099 {
8100   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8101   \@@_CodeAfter_iv:n
8102 }

```

We catch the argument of the command `\end` (in `#1`).

```

8103 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8104 {

```

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.

```

8105   \str_if_eq:eeTF \@currenvir { #1 }
8106   { \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`.

```

8107   {
8108     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8109     \@@_CodeAfter_ii:n
8110   }
8111 }

```

### 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.

```
8112 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8113 {
8114   \pgfpicture
8115   \pgfrememberpicturepositiononpagetrue
8116   \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.

```
8117   \@@_qpoint:n { row - 1 }
8118   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8119   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8120   \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).

```
8121   \bool_if:nTF { #3 }
8122   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8123   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8124   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8125   {
8126     \cs_if_exist:cT
8127     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8128     {
8129       \pgfpointanchor
8130       { \@@_env: - ##1 - #2 }
8131       { \bool_if:nTF { #3 } { west } { east } }
8132       \dim_set:Nn \l_tmpa_dim
8133       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8134     }
8135   }
```

Now we can put the delimiter with a node of PGF.

```
8136   \pgfset { inner~sep = \c_zero_dim }
8137   \dim_zero:N \nulldelimiterspace
8138   \pgftransformshift
8139   {
8140     \pgfpoint
8141     { \l_tmpa_dim }
8142     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8143   }
8144   \pgfnode
8145   { rectangle }
8146   { \bool_if:nTF { #3 } { east } { west } }
8147   {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
8148   \nullfont
8149   \c_math_toggle_token
8150   \@@_color:o \l_@@_delimiters_color_tl
8151   \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

8152     \vcenter
8153     {
8154         \nullfont
8155         \hrule \@height
8156             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8157             \@depth \c_zero_dim
8158             \@width \c_zero_dim
8159     }
8160     \bool_if:nTF { #3 } { \right . } { \right #1 }
8161     \c_math_toggle_token
8162 }
8163 { }
8164 { }
8165 \endpgfpicture
8166 }

```

## 34 The command \SubMatrix

```

8167 \keys_define:nn { NiceMatrix / sub-matrix }
8168 {
8169     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8170     extra-height .value_required:n = true ,
8171     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8172     left-xshift .value_required:n = true ,
8173     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8174     right-xshift .value_required:n = true ,
8175     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8176     xshift .value_required:n = true ,
8177     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8178     delimiters / color .value_required:n = true ,
8179     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8180     slim .default:n = true ,
8181     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8182     hlines .default:n = all ,
8183     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8184     vlines .default:n = all ,
8185     hvlines .meta:n = { hlines, vlines } ,
8186     hvlines .value_forbidden:n = true
8187 }
8188 \keys_define:nn { NiceMatrix }
8189 {
8190     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8191     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8192     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8193     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8194 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8195 \keys_define:nn { NiceMatrix / SubMatrix }
8196 {
8197     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8198     delimiters / color .value_required:n = true ,
8199     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8200     hlines .default:n = all ,
8201     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8202     vlines .default:n = all ,
8203     hvlines .meta:n = { hlines, vlines } ,
8204     hvlines .value_forbidden:n = true ,
8205     name .code:n =

```

```

8206 \tl_if_empty:nTF { #1 }
8207 { \@@_error:n { Invalid-name } }
8208 {
8209   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8210   {
8211     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8212     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8213     {
8214       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8215       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8216     }
8217   }
8218   { \@@_error:n { Invalid-name } }
8219 } ,
8220 name .value_required:n = true ,
8221 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8222 rules .value_required:n = true ,
8223 code .tl_set:N = \l_@@_code_tl ,
8224 code .value_required:n = true ,
8225 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8226 }

8227 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8228 {
8229   \peek_remove_spaces:n
8230   {
8231     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8232     {
8233       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8234       [
8235         delimiters / color = \l_@@_delimiters_color_tl ,
8236         hlines = \l_@@_submatrix_hlines_clist ,
8237         vlines = \l_@@_submatrix_vlines_clist ,
8238         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8239         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8240         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8241         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8242         #5
8243       ]
8244     }
8245     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8246   }
8247 }

8248 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8249 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8250 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8251 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8252 {
8253   \seq_gput_right:Nx \g_@@_submatrix_seq
8254   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8255 { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8256 { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8257 { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8258 { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8259 }
8260 }

```

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`.

```

8261 \hook_gput_code:nnn { begindocument } { . }
8262 {
8263   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8264   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8265   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8266     {
8267       \peek_remove_spaces:n
8268       {
8269         \@@_sub_matrix:nnnnnnn
8270         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8271       }
8272     }
8273 }

```

The following macro will compute `\l_@@_first_i_tl`, `\l_@@_first_j_tl`, `\l_@@_last_i_tl` and `\l_@@_last_j_tl` from the arguments of the command as provided by the user (for example 2-3 and 5-last).

```

8274 \NewDocumentCommand \@@_compute_i_j:nn
8275 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8276 { \@@_compute_i_j:nnnn #1 #2 }
8277 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8278 {
8279   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8280   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8281   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8282   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8283   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8284     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8285   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8286     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8287   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8288     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8289   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8290     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8291 }
8292 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8293 {
8294   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8295 \@@_compute_i_j:nn { #2 } { #3 }
8296 \int_compare:NnNT \l_@@_first_i_tl = \l_@@_last_i_tl
8297   { \cs_set_nopar:Npn \arraystretch { 1 } }
8298 \bool_lazy_or:nnTF
8299   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8300   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8301   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8302   {
8303     \str_clear_new:N \l_@@_submatrix_name_str
8304     \keys_set:nn { NiceMatrix / SubMatrix } { #5 }

```

```

8305 \pgfpicture
8306 \pgfrememberpicturepositiononpagetrue
8307 \pgf@relevantforpicturesizefalse
8308 \pgfset { inner~sep = \c_zero_dim }
8309 \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
8310 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currfication.

8311 \bool_if:NTF \l_@@_submatrix_slim_bool
8312 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8313 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8314 {
8315   \cs_if_exist:cT
8316   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8317   {
8318     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8319     \dim_set:Nn \l_@@_x_initial_dim
8320     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8321   }
8322   \cs_if_exist:cT
8323   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8324   {
8325     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8326     \dim_set:Nn \l_@@_x_final_dim
8327     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8328   }
8329 }
8330 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8331 { \@@_error:nn { Impossible~delimiter } { left } }
8332 {
8333   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8334   { \@@_error:nn { Impossible~delimiter } { right } }
8335   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8336 }
8337 \endpgfpicture
8338 }
8339 \group_end:
8340 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8341 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8342 {
8343   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8344   \dim_set:Nn \l_@@_y_initial_dim
8345   {
8346     \fp_to_dim:n
8347     {
8348       \pgf@y
8349       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8350     }
8351   }
8352   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8353   \dim_set:Nn \l_@@_y_final_dim
8354   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8355   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8356   {
8357     \cs_if_exist:cT
8358     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8359     {
8360       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8361       \dim_set:Nn \l_@@_y_initial_dim
8362       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8363     }

```

```

8364 \cs_if_exist:cT
8365 { pgf @ sh @ ns @ \l_@@_env: - \l_@@_last_i_tl - ##1 }
8366 {
8367 \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
8368 \dim_set:Nn \l_@@_y_final_dim
8369 { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8370 }
8371 }
8372 \dim_set:Nn \l_tmpa_dim
8373 {
8374 \l_@@_y_initial_dim - \l_@@_y_final_dim +
8375 \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8376 }
8377 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8378 \group_begin:
8379 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8380 \l_@@_set_CT@arc@:o \l_@@_rules_color_tl
8381 \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`.

```

8382 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8383 {
8384 \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8385 {
8386 \int_compare:nNnT
8387 { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8388 {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8389 \l_@@_qpoint:n { col - ##1 }
8390 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8391 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8392 \pgfusepathqstroke
8393 }
8394 }
8395 }

```

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.

```

8396 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8397 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8398 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8399 {
8400 \bool_lazy_and:nnTF
8401 { \int_compare_p:nNn { ##1 } > \c_zero_int }
8402 {
8403 \int_compare_p:nNn
8404 { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8405 {
8406 \l_@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8407 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8408 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8409 \pgfusepathqstroke
8410 }
8411 { \l_@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8412 }

```

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.

```

8413 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8414 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8415 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8416 {
8417   \bool_lazy_and:nnTF
8418   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8419   {
8420     \int_compare_p:nNn
8421     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8422   {
8423     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8424   \group_begin:
We compute in \l_tmpa_dim the  $x$ -value of the left end of the rule.
8425   \dim_set:Nn \l_tmpa_dim
8426   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8427   \str_case:nn { #1 }
8428   {
8429     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8430     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8431     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8432     }
8433   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8434   \dim_set:Nn \l_tmpb_dim
8435   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8436   \str_case:nn { #2 }
8437   {
8438     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8439     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8440     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8441   }
8442   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8443   \pgfusepathqstroke
8444   \group_end:
8445 }
8446 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8447 }

```

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).

```

8448 \str_if_empty:NF \l_@@_submatrix_name_str
8449 {
8450   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8451   \l_@@_x_initial_dim \l_@@_y_initial_dim
8452   \l_@@_x_final_dim \l_@@_y_final_dim
8453 }
8454 \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`.

```

8455 \begin { pgfscope }
8456 \pgftransformshift
8457 {
8458   \pgfpoint
8459   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8460   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8461 }
8462 \str_if_empty:NNTF \l_@@_submatrix_name_str
8463 { \@@_node_left:nn #1 { } }

```

```

8464     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8465 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8466 \pgftransformshift
8467 {
8468   \pgfpoint
8469   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8470   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8471 }
8472 \str_if_empty:NTF \l_@@_submatrix_name_str
8473 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8474 {
8475   \@@_node_right:nnnn #2
8476   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8477 }
8478 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8479 \flag_clear_new:n { nicematrix }
8480 \l_@@_code_tl
8481 }

```

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`.

```

8482 \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.

```

8483 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8484 {
8485   \use:e
8486   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8487 }

```

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 -.

```

8488 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8489 { #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.

```

8490 \tl_const:Nn \c_@@_integers_alist_tl
8491 {
8492   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8493   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8494   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8495   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8496 }
8497 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8498 {

```

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`.

```

8499 \tl_if_empty:nTF { #2 }
8500 {
8501   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8502   {
8503     \flag_raise:n { nicematrix }
8504     \int_if_even:nTF { \flag_height:n { nicematrix } }
8505     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8506     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8507   }
8508   { #1 }
8509 }

```

If there is an hyphen, we have to see whether we have a node of the form  $i-j$ ,  $\text{row-}i$  or  $\text{col-}j$ .

```

8510 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8511 }

```

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`).

```

8512 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8513 {
8514   \str_case:nnF { #1 }
8515   {
8516     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8517     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8518   }

```

Now the case of a node of the form  $i-j$ .

```

8519 {
8520   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8521   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8522 }
8523 }

```

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`).

```

8524 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8525 {
8526   \pgfnode
8527   { rectangle }
8528   { east }
8529   {
8530     \nullfont
8531     \c_math_toggle_token
8532     \@@_color:o \l_@@_delimiters_color_tl
8533     \left #1
8534     \vcenter
8535     {
8536       \nullfont
8537       \hrule \@height \l_tmpa_dim
8538       \c_zero_dim
8539       \c_zero_dim
8540     }
8541     \right .
8542     \c_math_toggle_token
8543   }
8544   { #2 }

```

```

8545     { }
8546 }

```

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.

```

8547 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8548 {
8549   \pgfnode
8550     { rectangle }
8551     { west }
8552     {
8553       \nullfont
8554       \c_math_toggle_token
8555       \@@_color:o \l_@@_delimiters_color_tl
8556       \left .
8557       \vcenter
8558         {
8559           \nullfont
8560           \hrule \@height \l_tmpa_dim
8561             \@depth \c_zero_dim
8562             \@width \c_zero_dim
8563         }
8564       \right #1
8565       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8566       ^ { \smash { #4 } }
8567       \c_math_toggle_token
8568     }
8569     { #2 }
8570     { }
8571 }

```

## 35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8572 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8573 {
8574   \peek_remove_spaces:n
8575   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8576 }
8577 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8578 {
8579   \peek_remove_spaces:n
8580   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8581 }
8582 \keys_define:nn { NiceMatrix / Brace }
8583 {
8584   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8585   left-shorten .default:n = true ,
8586   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8587   shorten .meta:n = { left-shorten , right-shorten } ,
8588   right-shorten .default:n = true ,
8589   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8590   yshift .value_required:n = true ,
8591   yshift .initial:n = \c_zero_dim ,
8592   color .tl_set:N = \l_tmpa_tl ,
8593   color .value_required:n = true ,

```

```

8594     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8595 }

```

#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`.

```

8596 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8597 {
8598     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8599     \@@_compute_i_j:nn { #1 } { #2 }
8600     \bool_lazy_or:nnTF
8601     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8602     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8603     {
8604         \str_if_eq:nnTF { #5 } { under }
8605         { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8606         { \@@_error:nn { Construct~too~large } { \OverBrace } }
8607     }
8608     {
8609         \tl_clear:N \l_tmpa_tl
8610         \keys_set:nn { NiceMatrix / Brace } { #4 }
8611         \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8612         \pgfpicture
8613         \pgfrememberpicturepositiononpagetrue
8614         \pgf@relevantforpicturesizefalse
8615         \bool_if:NT \l_@@_brace_left_shorten_bool
8616         {
8617             \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8618             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8619             {
8620                 \cs_if_exist:cT
8621                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8622                 {
8623                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8624                     \dim_set:Nn \l_@@_x_initial_dim
8625                     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8626                 }
8627             }
8628         }
8629         \bool_lazy_or:nnT
8630         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8631         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8632         {
8633             \@@_qpoint:n { col - \l_@@_first_j_tl }
8634             \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8635         }
8636         \bool_if:NT \l_@@_brace_right_shorten_bool
8637         {
8638             \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8639             \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8640             {
8641                 \cs_if_exist:cT
8642                 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8643                 {
8644                     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8645                     \dim_set:Nn \l_@@_x_final_dim
8646                     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8647                 }
8648             }
8649         }
8650         \bool_lazy_or:nnT
8651         { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```

```

8652         { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8653         {
8654             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8655             \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8656         }
8657         \pgfset { inner~sep = \c_zero_dim }
8658         \str_if_eq:nnTF { #5 } { under }
8659             { \@@_underbrace_i:n { #3 } }
8660             { \@@_overbrace_i:n { #3 } }
8661         \endpgfpicture
8662     }
8663 \group_end:
8664 }

```

The argument is the text to put above the brace.

```

8665 \cs_new_protected:Npn \@@_overbrace_i:n #1
8666 {
8667     \@@_qpoint:n { row - \l_@@_first_i_tl }
8668     \pgftransformshift
8669     {
8670         \pgfpoint
8671             { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8672             { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8673     }
8674     \pgfnode
8675     { rectangle }
8676     { south }
8677     {
8678         \vtop
8679         {
8680             \group_begin:
8681             \everycr { }
8682             \halign
8683             {
8684                 \hfil ## \hfil \crcr
8685                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8686                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8687                 \c_math_toggle_token
8688                 \overbrace
8689                 {
8690                     \hbox_to_wd:nn
8691                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8692                     { }
8693                 }
8694                 \c_math_toggle_token
8695                 \cr
8696             }
8697             \group_end:
8698         }
8699     }
8700     { }
8701     { }
8702 }

```

The argument is the text to put under the brace.

```

8703 \cs_new_protected:Npn \@@_underbrace_i:n #1
8704 {
8705     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8706     \pgftransformshift
8707     {
8708         \pgfpoint
8709             { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8710             { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```

```

8711     }
8712 \pgfnode
8713 { rectangle }
8714 { north }
8715 {
8716   \group_begin:
8717   \everycr { }
8718   \vbox
8719   {
8720     \halign
8721     {
8722       \hfil ## \hfil \crcr
8723       \c_math_toggle_token
8724       \underbrace
8725       {
8726         \hbox_to_wd:nn
8727         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8728         { }
8729       }
8730       \c_math_toggle_token
8731       \cr
8732       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8733       \@@_math_toggle: #1 \@@_math_toggle: \cr
8734     }
8735   }
8736   \group_end:
8737 }
8738 { }
8739 { }
8740 }

```

## 36 The command TikzEveryCell

```

8741 \bool_new:N \l_@@_not_empty_bool
8742 \bool_new:N \l_@@_empty_bool
8743
8744 \keys_define:nn { NiceMatrix / TikzEveryCell }
8745 {
8746   not-empty .code:n =
8747     \bool_lazy_or:nnTF
8748     \l_@@_in_code_after_bool
8749     \g_@@_recreate_cell_nodes_bool
8750     { \bool_set_true:N \l_@@_not_empty_bool }
8751     { \@@_error:n { detection~of~empty~cells } } ,
8752   not-empty .value_forbidden:n = true ,
8753   empty .code:n =
8754     \bool_lazy_or:nnTF
8755     \l_@@_in_code_after_bool
8756     \g_@@_recreate_cell_nodes_bool
8757     { \bool_set_true:N \l_@@_empty_bool }
8758     { \@@_error:n { detection~of~empty~cells } } ,
8759   empty .value_forbidden:n = true ,
8760   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8761 }
8762
8763
8764 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8765 {
8766   \IfPackageLoadedTF { tikz }

```

```

8767 {
8768     \group_begin:
8769     \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.

```

8770     \tl_set:Nn \l_tmpa_tl { { #2 } }
8771     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8772     { \@@_for_a_block:nnnnn ##1 }
8773     \@@_all_the_cells:
8774     \group_end:
8775 }
8776 { \@@_error:n { TikzEveryCell~without~tikz } }
8777 }
8778
8779 \tl_new:N \@@_i_tl
8780 \tl_new:N \@@_j_tl
8781
8782 \cs_new_protected:Nn \@@_all_the_cells:
8783 {
8784     \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8785     {
8786         \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8787         {
8788             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8789             {
8790                 \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8791                 { \@@_i_tl - \@@_j_tl }
8792                 {
8793                     \bool_set_false:N \l_tmpa_bool
8794                     \cs_if_exist:cTF
8795                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8796                     {
8797                         \bool_if:NF \l_@@_empty_bool
8798                         { \bool_set_true:N \l_tmpa_bool }
8799                     }
8800                     {
8801                         \bool_if:NF \l_@@_not_empty_bool
8802                         { \bool_set_true:N \l_tmpa_bool }
8803                     }
8804                     \bool_if:NT \l_tmpa_bool
8805                     {
8806                         \@@_block_tikz:nnnnV
8807                         \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8808                     }
8809                 }
8810             }
8811         }
8812     }
8813 }
8814
8815 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8816 {
8817     \bool_if:NF \l_@@_empty_bool
8818     {
8819         \@@_block_tikz:nnnnV
8820         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8821     }
8822     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8823 }
8824
8825 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8826 {
8827     \int_step_inline:nnn { #1 } { #3 }

```

```

8828 {
8829   \int_step_inline:nnn { #2 } { #4 }
8830   { \cs_set:cpn { cell - ##1 - #####1 } { } }
8831 }
8832 }

```

## 37 The command \ShowCellNames

```

8833 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8834 {
8835   \dim_zero_new:N \g_@@_tmpc_dim
8836   \dim_zero_new:N \g_@@_tmpd_dim
8837   \dim_zero_new:N \g_@@_tmpe_dim
8838   \int_step_inline:nn \c@iRow
8839   {
8840     \begin { pgfpicture }
8841     \@@_qpoint:n { row - ##1 }
8842     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8843     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8844     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8845     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8846     \bool_if:NTF \l_@@_in_code_after_bool
8847     \end { pgfpicture }
8848     \int_step_inline:nn \c@jCol
8849     {
8850       \hbox_set:Nn \l_tmpa_box
8851       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8852       \begin { pgfpicture }
8853       \@@_qpoint:n { col - #####1 }
8854       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8855       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8856       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8857       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8858       \endpgfpicture
8859       \end { pgfpicture }
8860       \fp_set:Nn \l_tmpa_fp
8861       {
8862         \fp_min:nn
8863         {
8864           \fp_min:nn
8865           {
8866             \dim_ratio:nn
8867             { \g_@@_tmpd_dim }
8868             { \box_wd:N \l_tmpa_box }
8869           }
8870           {
8871             \dim_ratio:nn
8872             { \g_tmpb_dim }
8873             { \box_ht_plus_dp:N \l_tmpa_box }
8874           }
8875         }
8876         { 1.0 }
8877       }
8878       \box_scale:Nnn \l_tmpa_box
8879       { \fp_use:N \l_tmpa_fp }
8880       { \fp_use:N \l_tmpa_fp }
8881       \pgfpicture
8882       \pgfrememberpicturepositiononpagetrue
8883       \pgf@relevantforpicturesizefalse
8884       \pgftransformshift
8885       {
8886         \pgfpoint

```

```

8887         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8888         { \dim_use:N \g_tmpa_dim }
8889     }
8890     \pgfnode
8891     { rectangle }
8892     { center }
8893     { \box_use:N \l_tmpa_box }
8894     { }
8895     { }
8896     \endpgfpicture
8897 }
8898 }
8899 }
8900 \NewDocumentCommand \@@_ShowCellNames { }
8901 {
8902     \bool_if:NT \l_@@_in_code_after_bool
8903     {
8904         \pgfpicture
8905         \pgfrememberpicturepositiononpagetrue
8906         \pgf@relevantforpicturesizefalse
8907         \pgfpathrectanglecorners
8908         { \@@_qpoint:n { 1 } }
8909         {
8910             \@@_qpoint:n
8911             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8912         }
8913         \pgfsetfillopacity { 0.75 }
8914         \pgfsetfillcolor { white }
8915         \pgfusepathqfill
8916         \endpgfpicture
8917     }
8918     \dim_zero_new:N \g_@@_tmpc_dim
8919     \dim_zero_new:N \g_@@_tmpd_dim
8920     \dim_zero_new:N \g_@@_tmpe_dim
8921     \int_step_inline:nn \c@iRow
8922     {
8923         \bool_if:NTF \l_@@_in_code_after_bool
8924         {
8925             \pgfpicture
8926             \pgfrememberpicturepositiononpagetrue
8927             \pgf@relevantforpicturesizefalse
8928         }
8929         { \begin { pgfpicture } }
8930         \@@_qpoint:n { row - ##1 }
8931         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8932         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8933         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8934         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8935         \bool_if:NTF \l_@@_in_code_after_bool
8936         { \endpgfpicture }
8937         { \end { pgfpicture } }
8938         \int_step_inline:nn \c@jCol
8939         {
8940             \hbox_set:Nn \l_tmpa_box
8941             {
8942                 \normalfont \Large \sffamily \bfseries
8943                 \bool_if:NTF \l_@@_in_code_after_bool
8944                 { \color { red } }
8945                 { \color { red ! 50 } }
8946                 ##1 - ###1
8947             }
8948             \bool_if:NTF \l_@@_in_code_after_bool
8949             {

```

```

8950         \pgfpicture
8951         \pgfrememberpicturerepositiononpagetrue
8952         \pgf@relevantforpicturesizefalse
8953     }
8954     { \begin { pgfpicture } }
8955     @@_qpoint:n { col - ####1 }
8956     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8957     @@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8958     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8959     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8960     \bool_if:NTF \l_@@_in_code_after_bool
8961     { \endpgfpicture }
8962     { \end { pgfpicture } }
8963     \fp_set:Nn \l_tmpa_fp
8964     {
8965         \fp_min:nn
8966         {
8967             \fp_min:nn
8968             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
8969             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
8970         }
8971         { 1.0 }
8972     }
8973     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8974     \pgfpicture
8975     \pgfrememberpicturerepositiononpagetrue
8976     \pgf@relevantforpicturesizefalse
8977     \pgftransformshift
8978     {
8979         \pgfpoint
8980         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8981         { \dim_use:N \g_tmpa_dim }
8982     }
8983     \pgfnode
8984     { rectangle }
8985     { center }
8986     { \box_use:N \l_tmpa_box }
8987     { }
8988     { }
8989     \endpgfpicture
8990 }
8991 }
8992 }

```

## 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.

```
8993 \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.

```

8994 \bool_new:N \g_@@_footnote_bool
8995 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
8996 {
8997     The~key~'\l_keys_key_str'~is~unknown. \\

```

```

8998     That~key~will~be~ignored. \\
8999     For~a~list~of~the~available~keys,~type~H~<return>.
9000 }
9001 {
9002     The~available~keys~are~(in~alphabetic~order):~
9003     footnote,~
9004     footnotehyper,~
9005     messages~for~Overleaf,~
9006     no~test~for~array,~
9007     renew~dots,~and~
9008     renew~matrix.
9009 }
9010 \keys_define:nn { NiceMatrix / Package }
9011 {
9012     renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9013     renew~dots .value_forbidden:n = true ,
9014     renew~matrix .code:n = \@@_renew_matrix: ,
9015     renew~matrix .value_forbidden:n = true ,
9016     messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9017     footnote .bool_set:N = \g_@@_footnote_bool ,
9018     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9019     no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
9020     no~test~for~array .default:n = true ,
9021     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9022 }
9023 \ProcessKeysOptions { NiceMatrix / Package }

9024 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9025 {
9026     You~can't~use~the~option~'footnote'~because~the~package~
9027     footnotehyper~has~already~been~loaded.~
9028     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9029     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9030     of~the~package~footnotehyper.\\
9031     The~package~footnote~won't~be~loaded.
9032 }
9033 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9034 {
9035     You~can't~use~the~option~'footnotehyper'~because~the~package~
9036     footnote~has~already~been~loaded.~
9037     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9038     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9039     of~the~package~footnote.\\
9040     The~package~footnotehyper~won't~be~loaded.
9041 }

9042 \bool_if:NT \g_@@_footnote_bool
9043 {
The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.
9044     \IfClassLoadedTF { beamer }
9045     { \bool_set_false:N \g_@@_footnote_bool }
9046     {
9047         \IfPackageLoadedTF { footnotehyper }
9048         { \@@_error:n { footnote~with~footnotehyper~package } }
9049         { \usepackage { footnote } }
9050     }
9051 }

9052 \bool_if:NT \g_@@_footnotehyper_bool
9053 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9054 \IfClassLoadedTF { beamer }
9055 { \bool_set_false:N \g_@@_footnote_bool }
9056 {
9057   \IfPackageLoadedTF { footnote }
9058   { \@@_error:n { footnotehyper~with~footnote~package } }
9059   { \usepackage { footnotehyper } }
9060 }
9061 \bool_set_true:N \g_@@_footnote_bool
9062 }

```

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.

```

9063 \bool_new:N \l_@@_underscore_loaded_bool
9064 \IfPackageLoadedTF { underscore }
9065 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9066 { }

9067 \hook_gput_code:nnn { begindocument } { . }
9068 {
9069   \bool_if:NF \l_@@_underscore_loaded_bool
9070   {
9071     \IfPackageLoadedTF { underscore }
9072     { \@@_error:n { underscore~after~nicematrix } }
9073     { }
9074   }
9075 }

```

## 40 Error messages of the package

```

9076 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9077 { \str_const:Nn \c_@@_available_keys_str { } }
9078 {
9079   \str_const:Nn \c_@@_available_keys_str
9080   { For~a~list~of~the~available~keys,~type~H~<return>. }
9081 }

9082 \seq_new:N \g_@@_types_of_matrix_seq
9083 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9084 {
9085   NiceMatrix ,
9086   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9087 }
9088 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9089 { \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`.

```

9090 \cs_new_protected:Npn \@@_error_too_much_cols:
9091 {
9092   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9093   {
9094     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9095     { \@@_fatal:n { too-much-cols-for-matrix } }
9096     {
9097       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9098       { \@@_fatal:n { too-much-cols-for-matrix } }
9099       {
9100         \bool_if:NF \l_@@_last_col_without_value_bool
9101         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9102       }
9103     }
9104   }
9105   { \@@_fatal:nn { too-much-cols-for-array } }
9106 }

```

The following command must *not* be protected since it's used in an error message.

```

9107 \cs_new:Npn \@@_message_hdotsfor:
9108 {
9109   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9110   { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
9111 }
9112 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9113 {
9114   Incompatible-options.\\
9115   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9116   The-output-will-not-be-reliable.
9117 }
9118 \@@_msg_new:nn { negative~weight }
9119 {
9120   Negative-weight.\\
9121   The-weight-of~the~'X'~columns~must~be~positive~and~you~have~used~
9122   the~value~'\int_use:N \l_@@_weight_int'.\\
9123   The~absolute~value~will~be~used.
9124 }
9125 \@@_msg_new:nn { last-col-not-used }
9126 {
9127   Column-not-used.\\
9128   The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9129   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9130 }
9131 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9132 {
9133   Too-much-columns.\\
9134   In~the~row~\int_eval:n { \c@iRow },~
9135   you~try~to~use~more~columns~
9136   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\\
9137   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9138   (plus~the~exterior~columns).~This~error~is~fatal.
9139 }
9140 \@@_msg_new:nn { too-much-cols-for-matrix }
9141 {
9142   Too-much-columns.\\
9143   In~the~row~\int_eval:n { \c@iRow },~
9144   you~try~to~use~more~columns~than~allowed~by~your~
9145   \@@_full_name_env:.~\@@_message_hdotsfor:\\ Recall~that~the~maximal~
9146   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9147   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~

```

```

9148     Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9149     \token_to_str:N \setcounter\ to~change~that~value).~
9150     This-error-is-fatal.
9151 }

9152 \@@_msg_new:nn { too-much-cols-for-array }
9153 {
9154     Too-much-columns.\\
9155     In~the~row~\int_eval:n { \c@iRow },~
9156     ~you~try~to~use~more~columns~than~allowed~by~your~
9157     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9158     \int_use:N \g_@@_static_num_of_col_int\
9159     ~(plus~the~potential~exterior~ones).
9160     This-error-is-fatal.
9161 }

9162 \@@_msg_new:nn { columns-not-used }
9163 {
9164     Columns-not-used.\\
9165     The-preamble-of~your~\@@_full_name_env:\ announces~\int_use:N
9166     \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9167     The-columns-you-did-not-used-won't-be-created.\\
9168     You-won't-have-similar-error-till-the-end-of-the-document.
9169 }

9170 \@@_msg_new:nn { in-first-col }
9171 {
9172     Erroneous-use.\\
9173     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9174     That-command-will-be-ignored.
9175 }

9176 \@@_msg_new:nn { in-last-col }
9177 {
9178     Erroneous-use.\\
9179     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9180     That-command-will-be-ignored.
9181 }

9182 \@@_msg_new:nn { in-first-row }
9183 {
9184     Erroneous-use.\\
9185     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9186     That-command-will-be-ignored.
9187 }

9188 \@@_msg_new:nn { in-last-row }
9189 {
9190     You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.\\
9191     That-command-will-be-ignored.
9192 }

9193 \@@_msg_new:nn { caption-outside-float }
9194 {
9195     Key-caption-forbidden.\\
9196     You-can't-use-the-key~'caption'~because~you-are~not~in~a~floating~
9197     environment.~This-key-will-be-ignored.
9198 }

9199 \@@_msg_new:nn { short-caption-without-caption }
9200 {
9201     You-should-not-use-the-key~'short-caption'~without~'caption'.~
9202     However,~your~'short-caption'~will-be-used-as~'caption'.
9203 }

9204 \@@_msg_new:nn { double-closing-delimiter }
9205 {
9206     Double-delimiter.\\

```

```

9207     You-can't-put-a-second-closing-delimiter-~"#1"~just-after-a-first-closing~
9208     delimiter.~This-delimiter-will-be-ignored.
9209 }
9210 \@@_msg_new:nn { delimiter-after-opening }
9211 {
9212     Double-delimiter.\\
9213     You-can't-put-a-second-delimiter-~"#1"~just-after-a-first-opening~
9214     delimiter.~That-delimiter-will-be-ignored.
9215 }
9216 \@@_msg_new:nn { bad-option-for-line-style }
9217 {
9218     Bad-line-style.\\
9219     Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to-'line-style'~
9220     is-'standard'.~That-key-will-be-ignored.
9221 }
9222 \@@_msg_new:nn { Identical-notes-in-caption }
9223 {
9224     Identical-tabular-notes.\\
9225     You-can't-put-several-notes-with-the-same-content-in~
9226     \token_to_str:N \caption\ (but-you-can-in-the-main-tabular).\\
9227     If-you-go-on,~the-output-will-probably-be-erroneous.
9228 }
9229 \@@_msg_new:nn { tabularnote-below-the-tabular }
9230 {
9231     \token_to_str:N \tabularnote\ forbidden\\
9232     You-can't-use-\token_to_str:N \tabularnote\ in-the-caption~
9233     of-your-tabular-because-the-caption-will-be-composed-below~
9234     the-tabular.~If-you-want-the-caption-above-the-tabular-use-the~
9235     key-'caption-above'~in-\token_to_str:N \NiceMatrixOptions.\\
9236     Your-\token_to_str:N \tabularnote\ will-be-discarded-and~
9237     no-similar-error-will-be-raised-in-this-document.
9238 }
9239 \@@_msg_new:nn { Unknown-key-for-rules }
9240 {
9241     Unknown-key.\\
9242     There-is-only-two-keys-available-here:~width-and-color.\\
9243     Your-key-'\l_keys_key_str'~will-be-ignored.
9244 }
9245 \@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
9246 {
9247     Unknown-key.\\
9248     There-is-only-two-keys-available-here:~
9249     'empty'~and~'not-empty'.\\
9250     Your-key-'\l_keys_key_str'~will-be-ignored.
9251 }
9252 \@@_msg_new:nn { Unknown-key-for-rotate }
9253 {
9254     Unknown-key.\\
9255     The-only-key-available-here-is-'c'.\\
9256     Your-key-'\l_keys_key_str'~will-be-ignored.
9257 }
9258 \@@_msg_new:nnn { Unknown-key-for-custom-line }
9259 {
9260     Unknown-key.\\
9261     The-key-'\l_keys_key_str'~is-unknown-in-a-'custom-line'.~
9262     It-you-go-on,~you-will-probably-have-other-errors. \\
9263     \c_@@_available_keys_str
9264 }
9265 {
9266     The-available-keys-are-(in-alphabetic-order):~

```

```

9267 ccommand,~
9268 color,~
9269 command,~
9270 dotted,~
9271 letter,~
9272 multiplicity,~
9273 sep-color,~
9274 tikz,~and~total-width.
9275 }
9276 \@@_msg_new:nnn { Unknown~key~for~xdots }
9277 {
9278   Unknown~key.\\
9279   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9280   \c_@@_available_keys_str
9281 }
9282 {
9283   The~available~keys~are~(in~alphabetic~order):~
9284   'color',~
9285   'horizontal-labels',~
9286   'inter',~
9287   'line-style',~
9288   'radius',~
9289   'shorten',~
9290   'shorten-end'~and~'shorten-start'.
9291 }
9292 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9293 {
9294   Unknown~key.\\
9295   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9296   (and~you~try~to~use~'\l_keys_key_str')\\
9297   That~key~will~be~ignored.
9298 }
9299 \@@_msg_new:nn { label~without~caption }
9300 {
9301   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9302   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9303 }
9304 \@@_msg_new:nn { W~warning }
9305 {
9306   Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9307   (row~\int_use:N \c@iRow).
9308 }
9309 \@@_msg_new:nn { Construct~too~large }
9310 {
9311   Construct~too~large.\\
9312   Your~command~\token_to_str:N #1
9313   can't~be~drawn~because~your~matrix~is~too~small.\\
9314   That~command~will~be~ignored.
9315 }
9316 \@@_msg_new:nn { underscore~after~nicematrix }
9317 {
9318   Problem~with~'underscore'.\\
9319   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9320   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9321   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9322 }
9323 \@@_msg_new:nn { ampersand~in~light-syntax }
9324 {
9325   Ampersand~forbidden.\\
9326   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9327   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.

```

```

9328 }
9329 \@@_msg_new:nn { double-backslash~in~light-syntax }
9330 {
9331   Double~backslash~forbidden.\\
9332   You~can't~use~\token_to_str:N
9333   \\~to~separate~rows~because~the~key~'light-syntax'~
9334   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9335   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9336 }
9337 \@@_msg_new:nn { hlines~with~color }
9338 {
9339   Incompatible~keys.\\
9340   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9341   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9342   Maybe~it~will~possible~in~future~version.\\
9343   Your~key~will~be~discarded.
9344 }
9345 \@@_msg_new:nn { bad~value~for~baseline }
9346 {
9347   Bad~value~for~baseline.\\
9348   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9349   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9350   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9351   the~form~'line-i'.\\
9352   A~value~of~1~will~be~used.
9353 }
9354 \@@_msg_new:nn { detection~of~empty~cells }
9355 {
9356   Problem~with~'not-empty'\\
9357   For~technical~reasons,~you~must~activate~
9358   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9359   in~order~to~use~the~key~'\l_keys_key_str'.\\
9360   That~key~will~be~ignored.
9361 }
9362 \@@_msg_new:nn { siunitx~not~loaded }
9363 {
9364   siunitx~not~loaded\\
9365   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9366   That~error~is~fatal.
9367 }
9368 \@@_msg_new:nn { ragged2e~not~loaded }
9369 {
9370   You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9371   your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9372   \l_keys_key_str'~will~be~used~instead.
9373 }
9374 \@@_msg_new:nn { Invalid~name }
9375 {
9376   Invalid~name.\\
9377   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9378   \SubMatrix\ of~your~\@@_full_name_env:.\
9379   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9380   This~key~will~be~ignored.
9381 }
9382 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9383 {
9384   Wrong~line.\\
9385   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9386   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9387   number~is~not~valid.~It~will~be~ignored.

```

```

9388 }
9389 \@@_msg_new:nn { Impossible-delimiter }
9390 {
9391   Impossible-delimiter.\\
9392   It's-impossible-to-draw-the-#1-delimiter-of-your-
9393   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty-
9394   in-that-column.
9395   \bool_if:NT \l_@@_submatrix_slim_bool
9396   { ~Maybe-you-should-try-without-the-key-'slim'. } \\
9397   This~\token_to_str:N \SubMatrix\ will-be-ignored.
9398 }
9399 \@@_msg_new:nnn { width-without-X-columns }
9400 {
9401   You-have-used-the-key-'width'~but-you-have-put-no-'X'~column.~
9402   That-key-will-be-ignored.
9403 }
9404 {
9405   This-message-is-the-message-'width-without-X-columns'~
9406   of-the-module-'nicematrix'.~
9407   The-experimented-users-can-disable-that-message-with-
9408   \token_to_str:N \msg_redirect_name:nnn.\\
9409 }
9410
9411 \@@_msg_new:nn { key-multiplicity-with-dotted }
9412 {
9413   Incompatible-keys. \\
9414   You-have-used-the-key-'multiplicity'~with-the-key-'dotted'~
9415   in-a-'custom-line'.~They-are-incompatible. \\
9416   The-key-'multiplicity'~will-be-discarded.
9417 }
9418 \@@_msg_new:nn { empty-environment }
9419 {
9420   Empty-environment.\\
9421   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9422 }
9423 \@@_msg_new:nn { No-letter-and-no-command }
9424 {
9425   Erroneous-use.\\
9426   Your-use-of-'custom-line'~is-no-op~since-you-don't-have-used-the-
9427   key-'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys-'command'~or~
9428   '~ccommand'~(to-draw-horizontal-rules).\\
9429   However,~you~can~go~on.
9430 }
9431 \@@_msg_new:nn { Forbidden-letter }
9432 {
9433   Forbidden-letter.\\
9434   You-can't-use-the-letter~'#1'~for-a-customized-line.\\
9435   It~will~be~ignored.
9436 }
9437 \@@_msg_new:nn { Several-letters }
9438 {
9439   Wrong-name.\\
9440   You-must-use-only-one-letter~as-value-for-the-key-'letter'~(and-you-
9441   have-used~'\l_@@_letter_str').\\
9442   It~will~be~ignored.
9443 }
9444 \@@_msg_new:nn { Delimiter-with-small }
9445 {
9446   Delimiter~forbidden.\\
9447   You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\

```

```

9448     because~the~key~'small'~is~in~force.\\
9449     This~error~is~fatal.
9450 }
9451 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9452 {
9453     Unknown~cell.\\
9454     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9455     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9456     can't~be~executed~because~a~cell~doesn't~exist.\\
9457     This~command~\token_to_str:N \line\ will~be~ignored.
9458 }
9459 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9460 {
9461     Duplicate~name.\\
9462     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9463     in~this~\@@_full_name_env:.\\
9464     This~key~will~be~ignored.\\
9465     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9466     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9467 }
9468 {
9469     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9470     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9471 }
9472 \@@_msg_new:nn { r~or~l~with~preamble }
9473 {
9474     Erroneous~use.\\
9475     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9476     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9477     your~\@@_full_name_env:.\\
9478     This~key~will~be~ignored.
9479 }
9480 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9481 {
9482     Erroneous~use.\\
9483     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9484     the~array.~This~error~is~fatal.
9485 }
9486 \@@_msg_new:nn { bad~corner }
9487 {
9488     Bad~corner.\\
9489     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9490     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9491     This~specification~of~corner~will~be~ignored.
9492 }
9493 \@@_msg_new:nn { bad~border }
9494 {
9495     Bad~border.\\
9496     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9497     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9498     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9499     also~use~the~key~'tikz'
9500     \IfPackageLoadedTF { tikz }
9501     { }
9502     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9503     This~specification~of~border~will~be~ignored.
9504 }
9505 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9506 {
9507     TikZ~not~loaded.\\
9508     You~can't~use~\token_to_str:N \TikzEveryCell\

```

```

9509     because~you~have~not~loaded~tikz.~
9510     This~command~will~be~ignored.
9511 }

9512 \@@_msg_new:nn { tikz-key-without~tikz }
9513 {
9514     TikZ~not~loaded.\\
9515     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9516     \Block'~because~you~have~not~loaded~tikz.~
9517     This~key~will~be~ignored.
9518 }

9519 \@@_msg_new:nn { last-col-non-empty~for~NiceArray }
9520 {
9521     Erroneous~use.\\
9522     In~the~\@@_full_name_env:,~you~must~use~the~key~
9523     'last-col'~without~value.\\
9524     However,~you~can~go~on~for~this~time~
9525     (the~value~'\l_keys_value_tl'~will~be~ignored).
9526 }

9527 \@@_msg_new:nn { last-col-non-empty~for~NiceMatrixOptions }
9528 {
9529     Erroneous~use.\\
9530     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9531     'last-col'~without~value.\\
9532     However,~you~can~go~on~for~this~time~
9533     (the~value~'\l_keys_value_tl'~will~be~ignored).
9534 }

9535 \@@_msg_new:nn { Block~too~large~1 }
9536 {
9537     Block~too~large.\\
9538     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9539     too~small~for~that~block. \\
9540     This~block~and~maybe~others~will~be~ignored.
9541 }

9542 \@@_msg_new:nn { Block~too~large~2 }
9543 {
9544     Block~too~large.\\
9545     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9546     \g_@@_static_num_of_col_int\
9547     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9548     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9549     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:~\\
9550     This~block~and~maybe~others~will~be~ignored.
9551 }

9552 \@@_msg_new:nn { unknown~column~type }
9553 {
9554     Bad~column~type.\\
9555     The~column~type~'#1'~in~your~\@@_full_name_env:\
9556     is~unknown. \\
9557     This~error~is~fatal.
9558 }

9559 \@@_msg_new:nn { unknown~column~type~S }
9560 {
9561     Bad~column~type.\\
9562     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9563     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9564     load~that~package. \\
9565     This~error~is~fatal.
9566 }

9567 \@@_msg_new:nn { tabularnote~forbidden }
9568 {

```

```

9569   Forbidden-command.\
9570   You-can't-use-the-command~\token_to_str:N\tabularnote\
9571   ~here~This-command-is-available-only-in~
9572   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9573   the-argument-of-a-command~\token_to_str:N \caption\ included~
9574   in-an-environment~{table}. \
9575   This-command-will-be-ignored.
9576 }

9577 \@@_msg_new:nn { borders~forbidden }
9578 {
9579   Forbidden-key.\
9580   You-can't-use-the-key~'borders'~of-the-command~\token_to_str:N \Block\
9581   because-the-option~'rounded-corners'~
9582   is-in-force-with-a-non-zero-value.\
9583   This-key-will-be-ignored.
9584 }

9585 \@@_msg_new:nn { bottomrule~without~booktabs }
9586 {
9587   booktabs-not-loaded.\
9588   You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
9589   loaded~'booktabs'.\
9590   This-key-will-be-ignored.
9591 }

9592 \@@_msg_new:nn { enumitem~not~loaded }
9593 {
9594   enumitem-not-loaded.\
9595   You-can't-use-the-command~\token_to_str:N\tabularnote\
9596   ~because-you-haven't~loaded~'enumitem'.\
9597   All-the-commands~\token_to_str:N\tabularnote\ will-be~
9598   ignored-in-the-document.
9599 }

9600 \@@_msg_new:nn { tikz~without~tikz }
9601 {
9602   Tikz-not-loaded.\
9603   You-can't-use-the-key~'tikz'~here~because-Tikz-is-not~
9604   loaded~If-you-go-on,~that-key-will-be-ignored.
9605 }

9606 \@@_msg_new:nn { tikz~in~custom-line~without~tikz }
9607 {
9608   Tikz-not-loaded.\
9609   You-have-used-the-key~'tikz'~in-the-definition-of-a~
9610   customized-line~(with~'custom-line')~but~tikz-is-not-loaded.~
9611   You-can-go-on-but-you-will-have-another-error-if-you-actually~
9612   use-that-custom-line.
9613 }

9614 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9615 {
9616   Tikz-not-loaded.\
9617   You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9618   command~'\token_to_str:N\Block')~but~tikz-is-not-loaded.~
9619   That-key-will-be-ignored.
9620 }

9621 \@@_msg_new:nn { without~color~inside }
9622 {
9623   If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9624   \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9625   outside~\token_to_str:N \CodeBefore,~you~
9626   should~have~used~the-key~'color~inside'~in-your~\@@_full_name_env:.\
9627   You-can-go-on-but-you-may~need~more~compilations.
9628 }

```

```

9629 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9630 {
9631   Erroneous-use.\
9632   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9633   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9634   The~key~'color'~will~be~discarded.
9635 }
9636 \@@_msg_new:nn { Wrong-last-row }
9637 {
9638   Wrong-number.\
9639   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9640   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9641   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9642   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9643   without~value~(more~compilations~might~be~necessary).
9644 }
9645 \@@_msg_new:nn { Yet-in-env }
9646 {
9647   Nested-environments.\
9648   Environments-of~nicematrix~can't~be~nested.\
9649   This~error~is~fatal.
9650 }
9651 \@@_msg_new:nn { Outside-math-mode }
9652 {
9653   Outside-math-mode.\
9654   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9655   (and~not~in~\token_to_str:N \vcenter).\
9656   This~error~is~fatal.
9657 }
9658 \@@_msg_new:nn { One-letter-allowed }
9659 {
9660   Bad-name.\
9661   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9662   It~will~be~ignored.
9663 }
9664 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9665 {
9666   Environment~{TabularNote}~forbidden.\
9667   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9668   but~*before*~the~\token_to_str:N \CodeAfter.\
9669   This~environment~{TabularNote}~will~be~ignored.
9670 }
9671 \@@_msg_new:nn { varwidth-not-loaded }
9672 {
9673   varwidth-not-loaded.\
9674   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9675   loaded.\
9676   Your~column~will~behave~like~'p'.
9677 }
9678 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9679 {
9680   Unknow-key.\
9681   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9682   \c_@@_available_keys_str
9683 }
9684 {
9685   The~available~keys~are~(in~alphabetic~order):~
9686   color,~
9687   dotted,~
9688   multiplicity,~
9689   sep-color,~

```

```

9690     tikz,~and~total~width.
9691 }
9692
9693 \@@_msg_new:nnn { Unknown~key~for~Block }
9694 {
9695     Unknown~key.\\
9696     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9697     \Block.\\ It~will~be~ignored. \\
9698     \c_@@_available_keys_str
9699 }
9700 {
9701     The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9702     hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9703     respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9704 }
9705 \@@_msg_new:nnn { Unknown~key~for~Brace }
9706 {
9707     Unknown~key.\\
9708     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9709     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9710     It~will~be~ignored. \\
9711     \c_@@_available_keys_str
9712 }
9713 {
9714     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9715     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9716     right~shorten)~and~yshift.
9717 }
9718 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9719 {
9720     Unknown~key.\\
9721     The~key~'\l_keys_key_str'~is~unknown.\\
9722     It~will~be~ignored. \\
9723     \c_@@_available_keys_str
9724 }
9725 {
9726     The~available~keys~are~(in~alphabetic~order):~
9727     delimiters/color,~
9728     rules~(with~the~subkeys~'color'~and~'width'),~
9729     sub~matrix~(several~subkeys)~
9730     and~xdots~(several~subkeys).~
9731     The~latter~is~for~the~command~\token_to_str:N \line.
9732 }
9733 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9734 {
9735     Unknown~key.\\
9736     The~key~'\l_keys_key_str'~is~unknown.\\
9737     It~will~be~ignored. \\
9738     \c_@@_available_keys_str
9739 }
9740 {
9741     The~available~keys~are~(in~alphabetic~order):~
9742     create~cell~nodes,~
9743     delimiters/color~and~
9744     sub~matrix~(several~subkeys).
9745 }
9746 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9747 {
9748     Unknown~key.\\
9749     The~key~'\l_keys_key_str'~is~unknown.\\
9750     That~key~will~be~ignored. \\
9751     \c_@@_available_keys_str

```

```

9752 }
9753 {
9754   The-available-keys-are~(in~alphabetic~order):~
9755   'delimiters/color',~
9756   'extra-height',~
9757   'hlines',~
9758   'hvlines',~
9759   'left-xshift',~
9760   'name',~
9761   'right-xshift',~
9762   'rules'~(with~the~subkeys~'color'~and~'width'),~
9763   'slim',~
9764   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9765   and~'right-xshift').\\
9766 }
9767 \@@_msg_new:nnn { Unknown~key~for~notes }
9768 {
9769   Unknown-key.\\
9770   The-key~'\l_keys_key_str'~is~unknown.\\
9771   That~key~will~be~ignored. \\
9772   \c_@@_available_keys_str
9773 }
9774 {
9775   The-available-keys-are~(in~alphabetic~order):~
9776   bottomrule,~
9777   code-after,~
9778   code-before,~
9779   detect-duplicates,~
9780   enumitem-keys,~
9781   enumitem-keys-para,~
9782   para,~
9783   label-in-list,~
9784   label-in-tabular~and~
9785   style.
9786 }
9787 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9788 {
9789   Unknown-key.\\
9790   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9791   \token_to_str:N \RowStyle. \\
9792   That~key~will~be~ignored. \\
9793   \c_@@_available_keys_str
9794 }
9795 {
9796   The-available-keys-are~(in~alphabetic~order):~
9797   'bold',~
9798   'cell-space-top-limit',~
9799   'cell-space-bottom-limit',~
9800   'cell-space-limits',~
9801   'color',~
9802   'nb-rows'~and~
9803   'rowcolor'.
9804 }
9805 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9806 {
9807   Unknown-key.\\
9808   The-key~'\l_keys_key_str'~is~unknown~for~the~command~
9809   \token_to_str:N \NiceMatrixOptions. \\
9810   That~key~will~be~ignored. \\
9811   \c_@@_available_keys_str
9812 }
9813 {
9814   The-available-keys-are~(in~alphabetic~order):~

```

```

9815 allow-duplicate-names,~
9816 caption-above,~
9817 cell-space-bottom-limit,~
9818 cell-space-limits,~
9819 cell-space-top-limit,~
9820 code-for-first-col,~
9821 code-for-first-row,~
9822 code-for-last-col,~
9823 code-for-last-row,~
9824 corners,~
9825 custom-key,~
9826 create-extra-nodes,~
9827 create-medium-nodes,~
9828 create-large-nodes,~
9829 delimiters~(several~subkeys),~
9830 end-of-row,~
9831 first-col,~
9832 first-row,~
9833 hlines,~
9834 hvlines,~
9835 hvlines-except-borders,~
9836 last-col,~
9837 last-row,~
9838 left-margin,~
9839 light-syntax,~
9840 matrix/columns-type,~
9841 no-cell-nodes,~
9842 notes~(several~subkeys),~
9843 nullify-dots,~
9844 pgf-node-code,~
9845 renew-dots,~
9846 renew-matrix,~
9847 respect-arraystretch,~
9848 rounded-corners,~
9849 right-margin,~
9850 rules~(with~the~subkeys~'color'~and~'width'),~
9851 small,~
9852 sub-matrix~(several~subkeys),~
9853 vlines,~
9854 xdots~(several~subkeys).
9855 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9856 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9857 {
9858   Unknown~key.\\
9859   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9860   \{NiceArray\}. \\
9861   That~key~will~be~ignored. \\
9862   \c_@@_available_keys_str
9863 }
9864 {
9865   The~available~keys~are~(in~alphabetic~order):~
9866   b,~
9867   baseline,~
9868   c,~
9869   cell-space-bottom-limit,~
9870   cell-space-limits,~
9871   cell-space-top-limit,~
9872   code-after,~
9873   code-for-first-col,~
9874   code-for-first-row,~
9875   code-for-last-col,~

```

```

9876 code-for-last-row,~
9877 color-inside,~
9878 columns-width,~
9879 corners,~
9880 create-extra-nodes,~
9881 create-medium-nodes,~
9882 create-large-nodes,~
9883 extra-left-margin,~
9884 extra-right-margin,~
9885 first-col,~
9886 first-row,~
9887 hlines,~
9888 hvlines,~
9889 hvlines-except-borders,~
9890 last-col,~
9891 last-row,~
9892 left-margin,~
9893 light-syntax,~
9894 name,~
9895 no-cell-nodes,~
9896 nullify-dots,~
9897 pgf-node-code,~
9898 renew-dots,~
9899 respect-arraystretch,~
9900 right-margin,~
9901 rounded-corners,~
9902 rules~(with~the~subkeys~'color'~and~'width'),~
9903 small,~
9904 t,~
9905 vlines,~
9906 xdots/color,~
9907 xdots/shorten-start,~
9908 xdots/shorten-end,~
9909 xdots/shorten-and~
9910 xdots/line-style.
9911 }

```

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).

```

9912 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9913 {
9914   Unknown~key.\\
9915   The~key~'\l_keys_key_str'~is~unknown~for~the~
9916   \@@_full_name_env:. \\
9917   That~key~will~be~ignored. \\
9918   \c_@@_available_keys_str
9919 }
9920 {
9921   The~available~keys~are~(in~alphabetic~order):~
9922   b,~
9923   baseline,~
9924   c,~
9925   cell-space-bottom-limit,~
9926   cell-space-limits,~
9927   cell-space-top-limit,~
9928   code-after,~
9929   code-for-first-col,~
9930   code-for-first-row,~
9931   code-for-last-col,~
9932   code-for-last-row,~
9933   color-inside,~
9934   columns-type,~
9935   columns-width,~
9936   corners,~

```

```

9937     create-extra-nodes,~
9938     create-medium-nodes,~
9939     create-large-nodes,~
9940     extra-left-margin,~
9941     extra-right-margin,~
9942     first-col,~
9943     first-row,~
9944     hlines,~
9945     hvlines,~
9946     hvlines-except-borders,~
9947     l,~
9948     last-col,~
9949     last-row,~
9950     left-margin,~
9951     light-syntax,~
9952     name,~
9953     no-cell-nodes,~
9954     nullify-dots,~
9955     pgf-node-code,~
9956     r,~
9957     renew-dots,~
9958     respect-arraystretch,~
9959     right-margin,~
9960     rounded-corners,~
9961     rules~(with~the~subkeys~'color'~and~'width'),~
9962     small,~
9963     t,~
9964     vlines,~
9965     xdots/color,~
9966     xdots/shorten-start,~
9967     xdots/shorten-end,~
9968     xdots/shorten-and~
9969     xdots/line-style.
9970 }
9971 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9972 {
9973     Unknown~key.\\
9974     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9975     \{NiceTabular\}. \\
9976     That~key~will~be~ignored. \\
9977     \c_@@_available_keys_str
9978 }
9979 {
9980     The~available~keys~are~(in~alphabetic~order):~
9981     b,~
9982     baseline,~
9983     c,~
9984     caption,~
9985     cell-space-bottom-limit,~
9986     cell-space-limits,~
9987     cell-space-top-limit,~
9988     code-after,~
9989     code-for-first-col,~
9990     code-for-first-row,~
9991     code-for-last-col,~
9992     code-for-last-row,~
9993     color-inside,~
9994     columns-width,~
9995     corners,~
9996     custom-line,~
9997     create-extra-nodes,~
9998     create-medium-nodes,~
9999     create-large-nodes,~

```

```

10000     extra-left-margin,~
10001     extra-right-margin,~
10002     first-col,~
10003     first-row,~
10004     hlines,~
10005     hvlines,~
10006     hvlines-except-borders,~
10007     label,~
10008     last-col,~
10009     last-row,~
10010     left-margin,~
10011     light-syntax,~
10012     name,~
10013     no-cell-nodes,~
10014     notes~(several~subkeys),~
10015     nullify-dots,~
10016     pgf-node-code,~
10017     renew-dots,~
10018     respect-arraystretch,~
10019     right-margin,~
10020     rounded-corners,~
10021     rules~(with~the~subkeys~'color'~and~'width'),~
10022     short-caption,~
10023     t,~
10024     tabularnote,~
10025     vlines,~
10026     xdots/color,~
10027     xdots/shorten-start,~
10028     xdots/shorten-end,~
10029     xdots/shorten-and~
10030     xdots/line-style.
10031 }

10032 \@@_msg_new:nnn { Duplicate-name }
10033 {
10034   Duplicate-name.\@
10035   The-name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10036   the~same~environment~name~twice.~You~can~go~on,~but,~
10037   maybe,~you~will~have~incorrect~results~especially~
10038   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10039   message~again,~use~the~key~'allow-duplicate-names'~in~
10040   '\token_to_str:N \NiceMatrixOptions'.\@
10041   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10042   { For~a~list~of~the~names~already~used,~type~H~<return>. }
10043 }
10044 {
10045   The~names~already~defined~in~this~document~are:~
10046   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10047 }

10048 \@@_msg_new:nn { Option-auto-for-columns-width }
10049 {
10050   Erroneous-use.\@
10051   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10052   That~key~will~be~ignored.
10053 }

10054 \@@_msg_new:nn { NiceTabularX~without~X }
10055 {
10056   NiceTabularX~without~X.\@
10057   You~should~not~use~{NiceTabularX}~without~X~columns.\@
10058   However,~you~can~go~on.
10059 }

10060 \@@_msg_new:nn { Preamble~forgotten }
10061 {

```

```
10062     Preamble~forgotten.\\
10063     You~have~probably~forgotten~the~preamble~of~your~
10064     \\@@_full_name_env:. \\
10065     This~error~is~fatal.
10066 }
```

# Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
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>	52
12	We construct the preamble of the array	57
13	The redefinition of <code>\multicolumn</code>	72
14	The environment <code>{NiceMatrix}</code> and its variants	90
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
16	After the construction of the array	92
17	We draw the dotted lines	99
18	The actual instructions for drawing the dotted lines with Tikz	112
19	User commands available in the new environments	117
20	The command <code>\line</code> accessible in code-after	123
21	The command <code>\RowStyle</code>	125
22	Colors of cells, rows and columns	127
23	The vertical and horizontal rules	140
24	The empty corners	154
25	The environment <code>{NiceMatrixBlock}</code>	157
26	The extra nodes	158
27	The blocks	163
28	How to draw the dotted lines transparently	182
29	Automatic arrays	183
30	The redefinition of the command <code>\dotfill</code>	184

31	The command <code>\diagbox</code>	184
32	The keyword <code>\CodeAfter</code>	186
33	The delimiters in the preamble	187
34	The command <code>\SubMatrix</code>	188
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	196
36	The command <code>TikzEveryCell</code>	199
37	The command <code>\ShowCellNames</code>	201
38	We process the options at package loading	203
39	About the package underscore	205
40	Error messages of the package	205