

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

May 6, 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`).

The development of the extension `nicematrix` is done on the following GitHub depot:  
<https://github.com/fpantigny/nicematrix>

## 1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

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

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

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

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

---

\*This document corresponds to the version 6.28 of `nicematrix`, at the date of 2024/04/30.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

## 2 Security test

Within the package `nicematrix`, we will have to test whether a cell of a `{NiceTabular}` is empty. For the cells of the columns of type `p`, `b`, `m`, `X` and `V`, we will test whether the cell is syntactically empty (that is to say that there is only spaces between the ampersands `&`). That test will be done with the command `\@@_test_if_empty`: by testing if the two first tokens in the cells are (during the TeX process) are `\ignorespaces` and `\unskip`.

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

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

```

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

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

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

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

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

```

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

```

96     }
97   }
98 }

```

### 3 Collecting options

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

*Exemple :*

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

Therefore, by writing : `\def\G{\@@_collect_options:n{\F}}`,  
the command `\G` takes in an arbitrary number of optional arguments between square brackets.  
Be careful: that command is *not* “fully expandable” (because of `\peek_meaning:NTF`).

```

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

```

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

```

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

```

### 4 Technical definitions

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

```

117 \tl_const:Nn \c_@@_b_tl { b }
118 \tl_const:Nn \c_@@_c_tl { c }
119 \tl_const:Nn \c_@@_l_tl { l }
120 \tl_const:Nn \c_@@_r_tl { r }
121 \tl_const:Nn \c_@@_all_tl { all }
122 \tl_const:Nn \c_@@_dot_tl { . }
123 \tl_const:Nn \c_@@_default_tl { default }
124 \tl_const:Nn \c_@@_star_tl { * }
125 \str_const:Nn \c_@@_r_str { r }
126 \str_const:Nn \c_@@_c_str { c }
127 \str_const:Nn \c_@@_l_str { l }
128 \str_const:Nn \c_@@_R_str { R }

```

```

129 \str_const:Nn \c_@@_C_str { C }
130 \str_const:Nn \c_@@_L_str { L }
131 \str_const:Nn \c_@@_j_str { j }
132 \str_const:Nn \c_@@_si_str { si }

```

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

```

133 \tl_new:N \l_@@_argspec_tl

134 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
135 \cs_generate_variant:Nn \str_lowercase:n { V }

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

```

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

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

```

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

```

We test whether the current class is `revtex4-1` (deprecated) or `revtex4-2` because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date April 2024, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

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

```

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

```

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

```

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

```

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

```

```

166     \ExplSyntaxOff
167   }
168   \cs_gset_eq:NN \@_provide_pgfsyspdfmark: \prg_do_nothing:
169 }

```

We define a command `\iddots` similar to `\ddots` (`\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.

```

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

```

This definition is a variant of the standard definition of `\ddots`.

In the `aux` file, we will have the references of the PGF/Tikz nodes created by `nicematrix`. However, when `booktabs` is used, some nodes (more precisely, some `row` nodes) will be defined twice because their position will be modified. In order to avoid an error message in this case, we will redefine `\pgfutil@check@rerun` in the `aux` file.

```

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

```

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

```

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

```

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

```

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

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. We will use it to store the instruction of color for the rules even if `colortbl` is not loaded.

```

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

```

```

209         { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
210     }

```

Idem for \CT@drs@.

```

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

```

We have to redefine \cline for several reasons. The command \@@\_cline will be linked to \cline in the beginning of {NiceArrayWithDelims}. The following commands must *not* be protected.

```

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

```

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

```

239     \skip_horizontal:N \c_zero_dim
240 }

```

Our \everycr has been modified. In particular, the creation of the row node is in the \everycr (maybe we should put it with the incrementation of \c@iRow). Since the following \cr correspond to a “false row”, we have to nullify \everycr.

```

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

```

The following version of \cline spreads the array of a quantity equal to \arrayrulewidth as does \hline. It will be loaded excepted if the key `standard-cline` has been used.

```

245 \cs_set:Npn \@@_cline

```

We have to act in a fully expandable way since there may be \noalign (in the \multispan) to detect. That’s why we use \@@\_cline\_i:en.

```

246 { \@@_cline_i:en \l_@@_first_col_int }

```

The command \cline\_i:nn has two arguments. The first is the number of the current column (it *must* be used in that column). The second is a standard argument of \cline of the form *i-j* or the form *i*.

---

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

```

247 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
248 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
249 {
250   \tl_if_empty:nTF { #3 }
251     { \@@_cline_iii:w #1|#2-#2 \q_stop }
252     { \@@_cline_ii:w #1|#2-#3 \q_stop }
253 }
254 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
255 { \@@_cline_iii:w #1|#2-#3 \q_stop }
256 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
257 {

```

Now, #1 is the number of the current column and we have to draw a line from the column #2 to the column #3 (both included).

```

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

```

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

```

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

```

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

```

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

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

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

```

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

```

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

```



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

```

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

299 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

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

```

## 5 Parameters

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

```

310 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

316 \bool_new:N \l_@@_tabular_bool

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

```
325 \dim_new:N \l_@@_col_width_dim
326 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
327 \int_new:N \g_@@_row_total_int
328 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
346 \bool_new:N \l_@@_X_bool
347 \bool_new:N \g_@@_caption_finished_bool
```

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

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

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

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

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

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

351 \tl_new:N \g_@@_left_delim_tl
352 \tl_new:N \g_@@_right_delim_tl
```

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

```
353 \tl_new:N \g_@@_user_preamble_tl
```

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

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

For `\multicolumn`.

```
355 \tl_new:N \g_@@_preamble_tl
```

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

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

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

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

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

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

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

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

```
368 \seq_new:N \g_@@_cols_vlism_seq
```

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

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

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

```
371 \str_new:N \g_@@_name_env_str
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
388 \int_new:N \l_@@_old_iRow_int
389 \int_new:N \l_@@_old_jCol_int
```

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

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

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

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

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

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

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

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

```
393 \bool_new:N \l_@@_X_columns_aux_bool
394 \dim_new:N \l_@@_X_columns_dim
```

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

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

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

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

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

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

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

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

```
398 \tl_new:N \l_@@_code_before_tl
399 \bool_new:N \l_@@_code_before_bool
```

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

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

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

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

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

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

407 \dim_new:N \g_@@_dp_row_zero_dim
408 \dim_new:N \g_@@_ht_row_zero_dim
409 \dim_new:N \g_@@_ht_row_one_dim
410 \dim_new:N \g_@@_dp_ante_last_row_dim
411 \dim_new:N \g_@@_ht_last_row_dim
412 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
413 \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”.

```
414 \dim_new:N \g_@@_width_last_col_dim
415 \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.

```
416 \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!).

```
417 \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}`.

```
418 \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.

```
419 \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.

```
420 \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`).

```
421 \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.

```
422 \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).

```
423 \seq_new:N \g_@@_multicolumn_cells_seq
424 \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`).

```
425 \int_new:N \l_@@_row_min_int
426 \int_new:N \l_@@_row_max_int
427 \int_new:N \l_@@_col_min_int
428 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
429 \int_new:N \l_@@_start_int
430 \int_set_eq:NN \l_@@_start_int \c_one_int
431 \int_new:N \l_@@_end_int
432 \int_new:N \l_@@_local_start_int
433 \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.

```
434 \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).

```
435 \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`.

```
436 \tl_new:N \l_@@_fill_tl
437 \tl_new:N \l_@@_opacity_tl
438 \tl_new:N \l_@@_draw_tl
439 \seq_new:N \l_@@_tikz_seq
440 \clist_new:N \l_@@_borders_clist
441 \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.

```
442 \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`.

```
443 \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.

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

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

```
445 \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`).

```
446 \str_new:N \l_@@_hpos_block_str
447 \str_set:Nn \l_@@_hpos_block_str { c }
448 \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.

```
449 \bool_new:N \l_@@_nocolor_used_bool
```

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

```
450 \str_new:N \l_@@_vpos_block_str
451 \str_set:Nn \l_@@_vpos_block_str { c }
```

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

```
452 \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).

```
453 \bool_new:N \l_@@_vlines_block_bool
454 \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.

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

456 \dim_new:N \l_@@_submatrix_extra_height_dim
457 \dim_new:N \l_@@_submatrix_left_xshift_dim
458 \dim_new:N \l_@@_submatrix_right_xshift_dim
459 \clist_new:N \l_@@_hlines_clist
460 \clist_new:N \l_@@_vlines_clist
461 \clist_new:N \l_@@_submatrix_hlines_clist
462 \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}).

```
463 \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.

```
464 \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).

```
465 \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.

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

```
468 \int_new:N \l_@@_first_col_int
469 \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).

```
470 \int_new:N \l_@@_last_row_int
471 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
473 \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`.

```
474 \int_new:N \l_@@_last_col_int
475 \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.

```
476 \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`).

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

## Some utilities

```
478 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
479 {
480   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
481   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
482 }
```

---

<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.

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.

```

483 \cs_new_protected:Npn \@@_expand_clist:N #1
484 {
485   \clist_if_in:NVF #1 \c_@@_all_tl
486   {
487     \clist_clear:N \l_tmpa_clist
488     \clist_map_inline:Nn #1
489     {
490       \tl_if_in:nnTF { ##1 } { - }
491       { \@@_cut_on_hyphen:w ##1 \q_stop }
492       {
493         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
494         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
495       }
496       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
497       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
498     }
499     \tl_set_eq:NN #1 \l_tmpa_clist
500   }
501 }

```

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.

```

502 \hook_gput_code:nnn { begindocument } { . }
503 {
504   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
505   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
506   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
507 }

```

## 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 tabulernote}`. The first component is the optional argument (between square brackets) of the command `\tabulernote` (if the optional argument is not used, the value will be the special marker expressed by `\c_novalue_tl`).
- 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 `tabulernote` 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.

```

508 \newcounter { tabulernote }
509 \seq_new:N \g_@@_notes_seq
510 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

511 \tl_new:N \g_@@_tabulernote_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.

```

512 \seq_new:N \l_@@_notes_labels_seq
513 \newcounter{nicematrix_draft}
514 \cs_new_protected:Npn \@@_notes_format:n #1
515 {
516   \setcounter { nicematrix_draft } { #1 }
517   \@@_notes_style:n { nicematrix_draft }
518 }

```

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

```

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

```

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

```

520 \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`.

```

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

```

We define `\thetabulernote` 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`.

```

522 \cs_set:Npn \thetabulernote { { \@@_notes_style:n { tabulernote } } }

```

---

<sup>3</sup>More precisely, it's the number of tabular notes which do not use the optional argument of `\tabulernote`.

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`).

```

523 \hook_gput_code:nnn { begindocument } { . }
524 {
525   \IfPackageLoadedTF { enumitem }
526   {

```

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.

```

527     \newlist { tabularnotes } { enumerate } { 1 }
528     \setlist [ tabularnotes ]
529     {
530       topsep = 0pt ,
531       noitemsep ,
532       leftmargin = * ,
533       align = left ,
534       labelsep = 0pt ,
535       label =
536         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
537     }
538     \newlist { tabularnotes* } { enumerate* } { 1 }
539     \setlist [ tabularnotes* ]
540     {
541       afterlabel = \nobreak ,
542       itemjoin = \quad ,
543       label =
544         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
545     }

```

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`.

```

546     \NewDocumentCommand \tabularnote { o m }
547     {
548       \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
549       {
550         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
551         { \@@_error:n { tabularnote~forbidden } }
552         {
553           \bool_if:NTF \l_@@_in_caption_bool
554             \@@_tabularnote_caption:nn
555             \@@_tabularnote:nn
556             { #1 } { #2 }
557         }
558       }
559     }
560   }
561   {
562     \NewDocumentCommand \tabularnote { o m }
563     {
564       \@@_error_or_warning:n { enumitem~not~loaded }
565       \@@_gredirect_none:n { enumitem~not~loaded }
566     }
567   }
568 }

569 \cs_new_protected:Npn \@@_test_first_novaluen { #1 #2 #3
570   { \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`.

```
571 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
572 {
```

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).

```
573   \int_zero:N \l_tmpa_int
574   \bool_if:NT \l_@@_notes_detect_duplicates_bool
575   {
```

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

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

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

```
576   \int_zero:N \l_tmpb_int
577   \seq_map_indexed_inline:Nn \g_@@_notes_seq
578   {
579     \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
580     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
581     {
582       \tl_if_novalue:nTF { #1 }
583       { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
584       { \int_set:Nn \l_tmpa_int { ##1 } }
585       \seq_map_break:
586     }
587   }
588   \int_if_zero:nF \l_tmpa_int
589   { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
590 }
591 \int_if_zero:nT \l_tmpa_int
592 {
593   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
594   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
595 }
596 \seq_put_right:Nx \l_@@_notes_labels_seq
597 {
598   \tl_if_novalue:nTF { #1 }
599   {
600     \@@_notes_format:n
601     {
602       \int_eval:n
603       {
604         \int_if_zero:nTF \l_tmpa_int
605         \c@tabularnote
606         \l_tmpa_int
607       }
608     }
609   }
610   { #1 }
611 }
612 \peek_meaning:NF \tabularnote
613 {
```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose

those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```
614 \hbox_set:Nn \l_tmpa_box
615 {
```

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

```
616 \@@_notes_label_in_tabular:n
617 {
618 \seq_use:Nnnn
619 \l_@@_notes_labels_seq { , } { , } { , }
620 }
621 }
```

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

```
622 \int_gdecr:N \c@tabularnote
623 \int_set_eq:NN \l_tmpa_int \c@tabularnote
624 \refstepcounter { tabularnote }
625 \int_compare:nNnT \l_tmpa_int = \c@tabularnote
626 { \int_gincr:N \c@tabularnote }
627 \seq_clear:N \l_@@_notes_labels_seq
628 \bool_lazy_or:nnTF
629 { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
630 { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
631 {
632 \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).

```
633 \skip_horizontal:n { \box_wd:N \l_tmpa_box }
634 }
635 { \box_use:N \l_tmpa_box }
636 }
637 }
```

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`.

```
638 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
639 {
640 \bool_if:NTF \g_@@_caption_finished_bool
641 {
642 \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
643 { \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`!

```
644 \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
645 { \@@_error:n { Identical-notes-in-caption } }
646 }
647 {
```

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

```
648 \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
649 {
```

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.

```

650         \bool_gset_true:N \g_@@_caption_finished_bool
651         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
652         \int_gzero:N \c@tabularnote
653     }
654     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
655 }

```

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!

```

656     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
657     \seq_put_right:Nx \l_@@_notes_labels_seq
658     {
659         \tl_if_novalue:nTF { #1 }
660         { \@@_notes_format:n { \int_use:N \c@tabularnote } }
661         { #1 }
662     }
663     \peek_meaning:NF \tabularnote
664     {
665         \@@_notes_label_in_tabular:n
666         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
667         \seq_clear:N \l_@@_notes_labels_seq
668     }
669 }

670 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
671 { \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.

```

672 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
673 {
674     \begin { pgfscope }
675     \pgfset
676     {
677         inner~sep = \c_zero_dim ,
678         minimum~size = \c_zero_dim
679     }
680     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
681     \pgfnode
682     { rectangle }
683     { center }
684     {
685         \vbox_to_ht:nn
686         { \dim_abs:n { #5 - #3 } }
687         {
688             \vfill
689             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
690         }
691     }
692     { #1 }
693     { }
694     \end { pgfscope }
695 }

```



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.

```

696 \cs_new_protected:Npn \@@pgf_rect_node:nnn #1 #2 #3
697 {
698   \begin { pgfscope }
699   \pgfset
700   {
701     inner~sep = \c_zero_dim ,
702     minimum~size = \c_zero_dim
703   }
704   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
705   \pgfpointdiff { #3 } { #2 }
706   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
707   \pgfnode
708   { rectangle }
709   { center }
710   {
711     \vbox_to_ht:nn
712     { \dim_abs:n \l_tmpb_dim }
713     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
714   }
715   { #1 }
716   { }
717   \end { pgfscope }
718 }

```

## 8 The options

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

```

719 \tl_new:N \l_@@_caption_tl
720 \tl_new:N \l_@@_short_caption_tl
721 \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).

```

722 \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.

```

723 \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`.

```

724 \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`).

```

725 \dim_new:N \l_@@_cell_space_top_limit_dim
726 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```
727 \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.

```
728 \dim_new:N \l_@@_xdots_inter_dim
729 \hook_gput_code:nnn { begindocument } { . }
730 { \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).

```
731 \dim_new:N \l_@@_xdots_shorten_start_dim
732 \dim_new:N \l_@@_xdots_shorten_end_dim
733 \hook_gput_code:nnn { begindocument } { . }
734 {
735   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
736   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
737 }
```

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.

```
738 \dim_new:N \l_@@_xdots_radius_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \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.

```
741 \tl_new:N \l_@@_xdots_line_style_tl
742 \tl_const:Nn \c_@@_standard_tl { standard }
743 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
744 \bool_new:N \l_@@_light_syntax_bool
745 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

```
746 \tl_new:N \l_@@_baseline_tl
747 \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`).

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

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

```
749 \bool_new:N \l_@@_parallelize_diags_bool
750 \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.

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

```
752 \dim_new:N \l_@@_notes_above_space_dim
```

```
753 \hook_gput_code:nnn { begindocument } { . }
```

```
754 { \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.).

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

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

```
756 \cs_new_protected:Npn \@@_reset_arraystretch:
```

```
757 { \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).

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

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

```
759 \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.

```
760 \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”.

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

```
762 \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).

```
763 \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`).

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

```
765 \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`.

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

```
767 \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.

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

```
769 \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 “:”.

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

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

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

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

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

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

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

```
811   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
812   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
813   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).

```

814     draw-first .code:n = \prg_do_nothing: ,
815     unknown .code:n = \@_error:n { Unknown~key~for~xdots }
816 }

```

```

817 \keys_define:nn { NiceMatrix / rules }
818 {
819     color .tl_set:N = \l_@@_rules_color_tl ,
820     color .value_required:n = true ,
821     width .dim_set:N = \arrayrulewidth ,
822     width .value_required:n = true ,
823     unknown .code:n = \@_error:n { Unknown~key~for~rules }
824 }

```

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

```

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

```

```

868 code-for-first-row .value_required:n = true ,
869 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
870 code-for-last-row .value_required:n = true ,
871 hlines .clist_set:N = \l_@@_hlines_clist ,
872 vlines .clist_set:N = \l_@@_vlines_clist ,
873 hlines .default:n = all ,
874 vlines .default:n = all ,
875 vlines-in-sub-matrix .code:n =
876 {
877   \tl_if_single_token:nTF { #1 }
878   {
879     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
880     { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

881       { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
882   }
883   { \@@_error:n { One~letter~allowed } }
884 } ,
885 vlines-in-sub-matrix .value_required:n = true ,
886 hvlines .code:n =
887 {
888   \bool_set_true:N \l_@@_hvlines_bool
889   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
890   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
891 } ,
892 hvlines-except-borders .code:n =
893 {
894   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
895   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
896   \bool_set_true:N \l_@@_hvlines_bool
897   \bool_set_true:N \l_@@_except_borders_bool
898 } ,
899 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.

```

900 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
901 renew-dots .value_forbidden:n = true ,
902 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
903 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
904 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
905 create-extra-nodes .meta:n =
906 { create-medium-nodes , create-large-nodes } ,
907 left-margin .dim_set:N = \l_@@_left_margin_dim ,
908 left-margin .default:n = \arraycolsep ,
909 right-margin .dim_set:N = \l_@@_right_margin_dim ,
910 right-margin .default:n = \arraycolsep ,
911 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
912 margin .default:n = \arraycolsep ,
913 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
914 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
915 extra-margin .meta:n =
916 { extra-left-margin = #1 , extra-right-margin = #1 } ,
917 extra-margin .value_required:n = true ,
918 respect-arraystretch .code:n =
919   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
920 respect-arraystretch .value_forbidden:n = true ,
921 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
922 pgf-node-code .value_required:n = true
923 }

```

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

```

924 \keys_define:nn { NiceMatrix / Env }
925 {
926   corners .clist_set:N = \l_@@_corners_clist ,
927   corners .default:n = { NW , SW , NE , SE } ,
928   code-before .code:n =
929   {
930     \tl_if_empty:nF { #1 }
931     {
932       \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
933       \bool_set_true:N \l_@@_code_before_bool
934     }
935   } ,
936   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}`.

```

937   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
938   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
939   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
940   baseline .tl_set:N = \l_@@_baseline_tl ,
941   baseline .value_required:n = true ,
942   columns-width .code:n =
943   { \tl_if_eq:nnTF { #1 } { auto }
944     { \bool_set_true:N \l_@@_auto_columns_width_bool }
945     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
946   columns-width .value_required:n = true ,
947   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.

```

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

```

```

978 label-in-tabular .value_required:n = true ,
979 label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
980 label-in-list .value_required:n = true ,
981 enumitem-keys .code:n =
982 {
983   \hook_gput_code:nnn { begindocument } { . }
984   {
985     \IfPackageLoadedTF { enumitem }
986       { \setlist* [ tabularnotes ] { #1 } }
987       { }
988   }
989 } ,
990 enumitem-keys .value_required:n = true ,
991 enumitem-keys-para .code:n =
992 {
993   \hook_gput_code:nnn { begindocument } { . }
994   {
995     \IfPackageLoadedTF { enumitem }
996       { \setlist* [ tabularnotes* ] { #1 } }
997       { }
998   }
999 } ,
1000 enumitem-keys-para .value_required:n = true ,
1001 detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1002 detect-duplicates .default:n = true ,
1003 unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1004 }
1005 \keys_define:nn { NiceMatrix / delimiters }
1006 {
1007   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1008   max-width .default:n = true ,
1009   color .tl_set:N = \l_@@_delimiters_color_tl ,
1010   color .value_required:n = true ,
1011 }

```

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

```

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

```



```

1037 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1038 NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1039 NiceArray .inherit:n =
1040 {
1041     NiceMatrix / Global ,
1042     NiceMatrix / Env ,
1043 } ,
1044 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1045 NiceArray / rules .inherit:n = NiceMatrix / rules ,
1046 pNiceArray .inherit:n =
1047 {
1048     NiceMatrix / Global ,
1049     NiceMatrix / Env ,
1050 } ,
1051 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1052 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1053 }

```

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

```

1054 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1055 {
1056     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1057     delimiters / color .value_required:n = true ,
1058     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1059     delimiters / max-width .default:n = true ,
1060     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1061     delimiters .value_required:n = true ,
1062     width .dim_set:N = \l_@@_width_dim ,
1063     width .value_required:n = true ,
1064     last-col .code:n =
1065     \tl_if_empty:nF { #1 }
1066     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1067     \int_zero:N \l_@@_last_col_int ,
1068     small .bool_set:N = \l_@@_small_bool ,
1069     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.

```

1070 renew-matrix .code:n = \@@_renew_matrix: ,
1071 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}`.

```

1072 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.

```

1073 columns-width .code:n =
1074 \tl_if_eq:nnTF { #1 } { auto }
1075 { \@@_error:n { Option-auto-for-columns-width } }
1076 { \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.

```

1077 allow-duplicate-names .code:n =
1078 \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1079 allow-duplicate-names .value_forbidden:n = true ,
1080 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1081 notes .value_required:n = true ,

```

```

1082   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1083   sub-matrix .value_required:n = true ,
1084   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1085   matrix / columns-type .value_required:n = true ,
1086   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1087   caption-above .default:n = true ,
1088   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrixOptions }
1089 }

```

`\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.

```

1090 \NewDocumentCommand \NiceMatrixOptions { m }
1091 { \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.

```

1092 \keys_define:nn { NiceMatrix / NiceMatrix }
1093 {
1094   last-col .code:n = \tl_if_empty:nTF { #1 }
1095   {
1096     \bool_set_true:N \l_@@_last_col_without_value_bool
1097     \int_set:Nn \l_@@_last_col_int { -1 }
1098   }
1099   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1100   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1101   columns-type .value_required:n = true ,
1102   l .meta:n = { columns-type = l } ,
1103   r .meta:n = { columns-type = r } ,
1104   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1105   delimiters / color .value_required:n = true ,
1106   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1107   delimiters / max-width .default:n = true ,
1108   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1109   delimiters .value_required:n = true ,
1110   small .bool_set:N = \l_@@_small_bool ,
1111   small .value_forbidden:n = true ,
1112   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1113 }

```

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

```

1114 \keys_define:nn { NiceMatrix / NiceArray }
1115 {

```

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.

```

1116   small .bool_set:N = \l_@@_small_bool ,
1117   small .value_forbidden:n = true ,
1118   last-col .code:n = \tl_if_empty:nF { #1 }
1119   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1120   \int_zero:N \l_@@_last_col_int ,
1121   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1122   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1123   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1124 }
1125 \keys_define:nn { NiceMatrix / pNiceArray }
1126 {
1127   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1128   last-col .code:n = \tl_if_empty:nF { #1 }
1129   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1130   \int_zero:N \l_@@_last_col_int ,

```

```

1131 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1132 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1133 delimiters / color .value_required:n = true ,
1134 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1135 delimiters / max-width .default:n = true ,
1136 delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1137 delimiters .value_required:n = true ,
1138 small .bool_set:N = \l_@@_small_bool ,
1139 small .value_forbidden:n = true ,
1140 r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1141 l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1142 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1143 }

```

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

```

1144 \keys_define:nn { NiceMatrix / NiceTabular }
1145 {

```

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.

```

1146 width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1147           \bool_set_true:N \l_@@_width_used_bool ,
1148 width .value_required:n = true ,
1149 notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1150 tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1151 tabularnote .value_required:n = true ,
1152 caption .tl_set:N = \l_@@_caption_tl ,
1153 caption .value_required:n = true ,
1154 short-caption .tl_set:N = \l_@@_short_caption_tl ,
1155 short-caption .value_required:n = true ,
1156 label .tl_set:N = \l_@@_label_tl ,
1157 label .value_required:n = true ,
1158 last-col .code:n = \tl_if_empty:nF {#1}
1159           { \@@_error:n { last-col-non-empty-for~NiceArray } }
1160           \int_zero:N \l_@@_last_col_int ,
1161 r .code:n = \@@_error:n { r-or~l-with~preamble } ,
1162 l .code:n = \@@_error:n { r-or~l-with~preamble } ,
1163 unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1164 }

```

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

```

1165 \keys_define:nn { NiceMatrix / CodeAfter }
1166 {
1167 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1168 delimiters / color .value_required:n = true ,
1169 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1170 rules .value_required:n = true ,
1171 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1172 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1173 sub-matrix .value_required:n = true ,
1174 unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1175 }

```

## 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}`).

```
1176 \cs_new_protected:Npn \@@_cell_begin:w
1177 {
```

`\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.

```
1178 \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).

```
1179 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1180 \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.

```
1181 \int_compare:nNnT \c@jCol = \c_one_int
1182 { \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:`.

```
1183 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1184 \@@_tuning_not_tabular_begin:
1185 \@@_tuning_first_row:
1186 \@@_tuning_last_row:
1187 \g_@@_row_style_tl
1188 }
```

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

```
1189 \cs_new_protected:Npn \@@_tuning_first_row:
1190 {
1191   \int_if_zero:nT \c@iRow
1192   {
1193     \int_compare:nNnT \c@jCol > \c_zero_int
1194     {
1195       \l_@@_code_for_first_row_tl
1196       \xglobal \colorlet { nicematrix-first-row } { . }
1197     }
1198   }
1199 }
```

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

```
1200 \cs_new_protected:Npn \@@_tuning_last_row:
1201 {
1202   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1203   {
1204     \l_@@_code_for_last_row_tl
1205     \xglobal \colorlet { nicematrix-last-row } { . }
1206   }
1207 }
```

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

```
1208 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1209 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1210 {
1211   \c_math_toggle_token
```

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

```
1212   \@@_tuning_key_small:
1213 }
1214 \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.

```
1215 \cs_new_protected:Npn \@@_begin_of_row:
1216 {
1217   \int_gincr:N \c@iRow
1218   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1219   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1220   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1221   \pgfpicture
1222   \pgfrememberpicturepositiononpagetrue
1223   \pgfcoordinate
1224     { \@@_env: - row - \int_use:N \c@iRow - base }
1225     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1226   \str_if_empty:NF \l_@@_name_str
1227     {
1228       \pgfnodealias
1229         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1230         { \@@_env: - row - \int_use:N \c@iRow - base }
1231     }
1232   \endpgfpicture
1233 }
```

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.

```
1234 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1235 {
1236   \int_if_zero:nTF \c@iRow
1237     {
1238     \dim_gset:Nn \g_@@_dp_row_zero_dim
1239       { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1240     \dim_gset:Nn \g_@@_ht_row_zero_dim
1241       { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1242     }
1243     {
1244     \int_compare:nNnT \c@iRow = \c_one_int
1245       {
1246       \dim_gset:Nn \g_@@_ht_row_one_dim
1247         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1248       }
1249     }
1250 }

1251 \cs_new_protected:Npn \@@_rotate_cell_box:
1252 {
1253   \box_rotate:Nn \l_@@_cell_box { 90 }
1254   \bool_if:NTF \g_@@_rotate_c_bool
1255     {
1256     \hbox_set:Nn \l_@@_cell_box
```

```

1257     {
1258         \c_math_toggle_token
1259         \vcenter { \box_use:N \l_@@_cell_box }
1260         \c_math_toggle_token
1261     }
1262 }
1263 {
1264     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1265     {
1266         \vbox_set_top:Nn \l_@@_cell_box
1267         {
1268             \vbox_to_zero:n { }
1269             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1270             \box_use:N \l_@@_cell_box
1271         }
1272     }
1273 }
1274 \bool_gset_false:N \g_@@_rotate_bool
1275 \bool_gset_false:N \g_@@_rotate_c_bool
1276 }
1277 \cs_new_protected:Npn \@@_adjust_size_box:
1278 {
1279     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1280     {
1281         \box_set_wd:Nn \l_@@_cell_box
1282         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1283         \dim_gzero:N \g_@@_blocks_wd_dim
1284     }
1285     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1286     {
1287         \box_set_dp:Nn \l_@@_cell_box
1288         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1289         \dim_gzero:N \g_@@_blocks_dp_dim
1290     }
1291     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1292     {
1293         \box_set_ht:Nn \l_@@_cell_box
1294         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1295         \dim_gzero:N \g_@@_blocks_ht_dim
1296     }
1297 }
1298 \cs_new_protected:Npn \@@_cell_end:
1299 {

```

The following command is nullified in the tabulars.

```

1300     \@@_tuning_not_tabular_end:
1301     \hbox_set_end:
1302     \@@_cell_end_i:
1303 }
1304 \cs_new_protected:Npn \@@_cell_end_i:
1305 {

```

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.

```

1306     \g_@@_cell_after_hook_tl
1307     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1308     \@@_adjust_size_box:
1309     \box_set_ht:Nn \l_@@_cell_box
1310     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1311     \box_set_dp:Nn \l_@@_cell_box
1312     { \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”).

```
1313 \@@_update_max_cell_width:
```

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

```
1314 \@@_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.

```
1315 \bool_if:NTF \g_@@_empty_cell_bool
1316 { \box_use_drop:N \l_@@_cell_box }
1317 {
1318   \bool_if:NTF \g_@@_not_empty_cell_bool
1319   \@@_node_for_cell:
1320   {
1321     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1322     \@@_node_for_cell:
1323     { \box_use_drop:N \l_@@_cell_box }
1324   }
1325 }
1326 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1327 \bool_gset_false:N \g_@@_empty_cell_bool
1328 \bool_gset_false:N \g_@@_not_empty_cell_bool
1329 }
```

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

```
1330 \cs_new_protected:Npn \@@_update_max_cell_width:
1331 {
1332   \dim_gset:Nn \g_@@_max_cell_width_dim
1333   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1334 }
```

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`).

```
1335 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1336 {
1337   \@@_math_toggle:
1338   \hbox_set_end:
1339   \bool_if:NF \g_@@_rotate_bool
1340   {
1341     \hbox_set:Nn \l_@@_cell_box
1342     {
```

```

1343         \makebox [ \l_@@_col_width_dim ] [ s ]
1344         { \hbox_unpack_drop:N \l_@@_cell_box }
1345     }
1346 }
1347 \@@_cell_end_i:
1348 }

1349 \pgfset
1350 {
1351     nicematrix / cell-node /.style =
1352     {
1353         inner~sep = \c_zero_dim ,
1354         minimum~width = \c_zero_dim
1355     }
1356 }

```

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

```

1357 \cs_new_protected:Npn \@@_node_for_cell:
1358 {
1359     \pgfpicture
1360     \pgfsetbaseline \c_zero_dim
1361     \pgfrememberpicturepositiononpagetrue
1362     \pgfset { nicematrix / cell-node }
1363     \pgfnode
1364     { rectangle }
1365     { base }
1366     {

```

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

```

1367         \set@color
1368         \box_use_drop:N \l_@@_cell_box
1369     }
1370     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1371     { \l_@@_pgf_node_code_tl }
1372     \str_if_empty:NF \l_@@_name_str
1373     {
1374         \pgfnodealias
1375         { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1376         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1377     }
1378     \endpgfpicture
1379 }

```

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.

```

1380 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1381 {
1382     \cs_new_protected:Npn \@@_patch_node_for_cell:
1383     {
1384         \hbox_set:Nn \l_@@_cell_box
1385         {
1386             \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1387             \hbox_overlap_left:n
1388             {
1389                 \pgfsys@markposition
1390                 { \@@_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.

```

1391         #1
1392     }
1393     \box_use:N \l_@@_cell_box
1394     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1395     \hbox_overlap_left:n
1396     {
1397         \pgfsys@markposition
1398         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1399         #1
1400     }
1401 }
1402 }
1403 }

```

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

```

1404 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1405 {
1406     \@@_patch_node_for_cell:n
1407     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1408 }
1409 { \@@_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).

```

1410 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1411 {
1412     \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1413     { g_@@_ #2 _ lines _ tl }
1414     {
1415         \use:c { @@ _ draw _ #2 : nnn }
1416         { \int_use:N \c@iRow }
1417         { \int_use:N \c@jCol }
1418         { \exp_not:n { #3 } }
1419     }
1420 }

1421 \cs_new_protected:Npn \@@_array:
1422 {
1423     % \begin{macrocode}
1424     \dim_set:Nn \col@sep
1425     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }

```

```

1426 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1427 { \cs_set_nopar:Npn \@halignto { } }
1428 { \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.

```

1429 \@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.

```

1430 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1431 }

```

We keep in memory the standard version of \ialign because we will redefine \ialign in the environment {NiceArrayWithDelims} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), array uses \ar@ialign instead of \ialign. In that case, of course, you do a saving of \ar@ialign.

```

1432 \bool_if:NNTF \c_@@_tagging_array_bool
1433 { \cs_set_eq:NN \@_old_ar@ialign: \ar@ialign }
1434 { \cs_set_eq:NN \@_old_ialign: \ialign }

```

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

```

1435 \cs_new_protected:Npn \@_create_row_node:
1436 {
1437   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1438   {
1439     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1440     \@_create_row_node_i:
1441   }
1442 }
1443 \cs_new_protected:Npn \@_create_row_node_i:
1444 {

```

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

```

1445 \hbox
1446 {
1447   \bool_if:NT \l_@@_code_before_bool
1448   {
1449     \vtop
1450     {
1451       \skip_vertical:N 0.5\arrayrulewidth
1452       \pgfsys@markposition
1453       { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1454       \skip_vertical:N -0.5\arrayrulewidth
1455     }
1456   }
1457   \pgfpicture
1458   \pgfrememberpicturepositiononpagetrue
1459   \pgfcoordinate { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1460   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1461   \str_if_empty:NF \l_@@_name_str
1462   {
1463     \pgfnodealias
1464     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1465     { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1466   }
1467   \endpgfpicture
1468 }
1469 }

```

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

```

1470 \cs_new:Npn \@_everycr: { \noalign { \@_everycr_i: } }

```

```

1471 \cs_new_protected:Npn \@@_everycr_i:
1472 {
1473   \bool_if:NT \c_@@_tagging_array_bool
1474   {
1475     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1476     \tbl_update_cell_data_for_next_row:
1477   }
1478   \int_gzero:N \c@jCol
1479   \bool_gset_false:N \g_@@_after_col_zero_bool
1480   \bool_if:NF \g_@@_row_of_col_done_bool
1481   {
1482     \@@_create_row_node:

```

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

```

1483     \tl_if_empty:NF \l_@@_hlines_clist
1484     {
1485       \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1486       {
1487         \exp_args:NNe
1488         \clist_if_in:NnT
1489         \l_@@_hlines_clist
1490         { \int_eval:n { \c@iRow + 1 } }
1491       }
1492     }

```

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.

```

1493     \int_compare:nNnT \c@iRow > { -1 }
1494     {
1495       \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@`.

```

1496         { \hrule height \arrayrulewidth width \c_zero_dim }
1497       }
1498     }
1499   }
1500 }
1501 }

```

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

```

1502 \cs_set_protected:Npn \@@_renew_dots:
1503 {
1504   \cs_set_eq:NN \ldots \@@_Ldots
1505   \cs_set_eq:NN \cdots \@@_Cdots
1506   \cs_set_eq:NN \vdots \@@_Vdots
1507   \cs_set_eq:NN \ddots \@@_Ddots
1508   \cs_set_eq:NN \iddots \@@_Iddots
1509   \cs_set_eq:NN \dots \@@_Ldots
1510   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1511 }
1512 \cs_new_protected:Npn \@@_test_color_inside:
1513 {
1514   \bool_if:NF \l_@@_color_inside_bool
1515   {

```

We will issue an error only during the first run.

```

1516     \bool_if:NF \g_@@_aux_found_bool
1517     { \@@_error:n { without~color~inside } }
1518   }
1519 }

```

```

1520 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1521 \hook_gput_code:nnn { begindocument } { . }
1522 {
1523   \IfPackageLoadedTF { colortbl }
1524   {
1525     \cs_set_protected:Npn \@@_redefine_everycr:
1526     {
1527       \CT@everycr
1528       {
1529         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1530         \@@_everycr:
1531       }
1532     }
1533   }
1534   { }
1535 }

```

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

```

1536 \hook_gput_code:nnn { begindocument } { . }
1537 {
1538   \IfPackageLoadedTF { booktabs }
1539   {
1540     \cs_new_protected:Npn \@@_patch_booktabs:
1541     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1542   }
1543   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1544 }

```

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`.

```

1545 \cs_new_protected:Npn \@@_some_initialization:
1546 {
1547   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1548   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1549   \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1550   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1551   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1552   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1553 }

```

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

```

1554 \cs_new_protected:Npn \@@_pre_array_ii:
1555 {

```

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

```

1556   \int_gzero:N \g_@@_total_X_weight_int

```

---

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

<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}`.

```

1557 \@@_expand_clist:N \l_@@_hlines_clist
1558 \@@_expand_clist:N \l_@@_vlines_clist
1559 \@@_patch_booktabs:
1560 \box_clear_new:N \l_@@_cell_box
1561 \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}`).

```

1562 \bool_if:NT \l_@@_small_bool
1563 {
1564     \cs_set_nopar:Npn \arraystretch { 0.47 }
1565     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1566     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1567 }

```

```

1568 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1569 {
1570     \tl_put_right:Nn \@@_begin_of_row:
1571     {
1572         \pgfsys@markposition
1573         { \@@_env: - row - \int_use:N \c@iRow - base }
1574     }
1575 }

```

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`.

```

1576 \bool_if:NTF \c_@@_tagging_array_bool
1577 {
1578     \cs_set_nopar:Npn \ar@ialign
1579     {
1580         \tbl_init_cell_data_for_table:
1581         \@@_redefine_everycr:
1582         \tabskip = \c_zero_skip
1583         \@@_some_initialization:

```

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

```

1584         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1585         \halign
1586     }
1587 }

```

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

```

1588 {
1589     \cs_set_nopar:Npn \ialign
1590     {
1591         \@@_redefine_everycr:
1592         \tabskip = \c_zero_skip
1593         \@@_some_initialization:
1594         \cs_set_eq:NN \ialign \@@_old_ialign:
1595         \halign
1596     }
1597 }

```

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).

```

1598 \cs_set_eq:NN \@@_old_ldots \ldots
1599 \cs_set_eq:NN \@@_old_cdots \cdots
1600 \cs_set_eq:NN \@@_old_vdots \vdots
1601 \cs_set_eq:NN \@@_old_ddots \ddots
1602 \cs_set_eq:NN \@@_old_iddots \iddots
1603 \bool_if:NTF \l_@@_standard_cline_bool
1604 { \cs_set_eq:NN \cline \@@_standard_cline }
1605 { \cs_set_eq:NN \cline \@@_cline }
1606 \cs_set_eq:NN \Ldots \@@_Ldots
1607 \cs_set_eq:NN \Cdots \@@_Cdots
1608 \cs_set_eq:NN \Vdots \@@_Vdots
1609 \cs_set_eq:NN \Ddots \@@_Ddots
1610 \cs_set_eq:NN \Iddots \@@_Iddots
1611 \cs_set_eq:NN \Hline \@@_Hline:
1612 \cs_set_eq:NN \Hspace \@@_Hspace:
1613 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1614 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1615 \cs_set_eq:NN \Block \@@_Block:
1616 \cs_set_eq:NN \rotate \@@_rotate:
1617 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1618 \cs_set_eq:NN \dotfill \@@_dotfill:
1619 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1620 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1621 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1622 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1623 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1624 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1625 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1626 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1627 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1628 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1629 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1630 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1631 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1632 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1633 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

1634 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1635 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1636 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1637 \@@_revert_colortbl:

```

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

```

1638 \tl_if_exist:NT \l_@@_note_in_caption_tl
1639 {
1640   \tl_if_empty:NF \l_@@_note_in_caption_tl
1641   {
1642     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1643     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1644   }
1645 }

```

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).

```
1646 \seq_gclear:N \g_@@_multicolumn_cells_seq
1647 \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).

```
1648 \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`).

```
1649 \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.

```
1650 \int_gzero_new:N \g_@@_col_total_int
1651 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1652 \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.

```
1653 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1654 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1655 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1656 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1657 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1658 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1659 \tl_gclear:N \g_nicematrix_code_before_tl
1660 \tl_gclear:N \g_@@_pre_code_before_tl
1661 }
```

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

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

```
1662 \cs_new_protected:Npn \@@_pre_array:
1663 {
1664   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1665   \int_gzero_new:N \c@iRow
1666   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1667   \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).

```
1668 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1669 {
1670   \bool_set_true:N \l_@@_last_row_without_value_bool
1671   \bool_if:NT \g_@@_aux_found_bool
1672     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1673 }
1674 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1675 {
1676   \bool_if:NT \g_@@_aux_found_bool
1677     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1678 }
```

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”.

```

1679 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1680 {
1681   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1682   {
1683     \dim_gset:Nn \g_@@_ht_last_row_dim
1684     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1685     \dim_gset:Nn \g_@@_dp_last_row_dim
1686     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1687   }
1688 }

1689 \seq_gclear:N \g_@@_cols_vlism_seq
1690 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1691 \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.

```

1692 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1693 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1694 \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.

```

1695 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1696 \@@_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.

```

1697 \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]`).

```

1698 \dim_zero_new:N \l_@@_left_delim_dim
1699 \dim_zero_new:N \l_@@_right_delim_dim
1700 \bool_if:NTF \g_@@_delims_bool
1701 {

```

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

```

1702   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1703   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1704   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1705   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1706 }
1707 {
1708   \dim_gset:Nn \l_@@_left_delim_dim
1709   { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1710   \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1711 }

```



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).

```

1712 \hbox_set:Nw \l_@@_the_array_box
1713 \bool_if:NT \c_@@_tagging_array_bool
1714 { \UseTaggingSocket { tbl / hmode / begin } }

1715 \skip_horizontal:N \l_@@_left_margin_dim
1716 \skip_horizontal:N \l_@@_extra_left_margin_dim
1717 \c_math_toggle_token
1718 \bool_if:NTF \l_@@_light_syntax_bool
1719 { \use:c { @@-light-syntax } }
1720 { \use:c { @@-normal-syntax } }
1721 }

```

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

```

1722 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1723 {
1724   \tl_set:Nn \l_tmpa_tl { #1 }
1725   \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
1726   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1727   \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1728   \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...

```

1729 \@@_pre_array:
1730 }

```

## 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).

```

1731 \cs_new_protected:Npn \@@_pre_code_before:
1732 {

```

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).

```

1733 \int_set:Nn \c_iRow { \seq_item:Nn \g_@@_size_seq 2 }
1734 \int_set:Nn \c_jCol { \seq_item:Nn \g_@@_size_seq 5 }
1735 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1736 \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.

```

1737 \pgfsys@markposition { \@@_env: - position }
1738 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1739 \pgfpicture
1740 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1741 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1742 {
1743   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1744   \pgfcoordinate { \@@_env: - row - ##1 }
1745   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1746 }

```

Now, the recreation of the col nodes.

```

1747 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1748 {
1749   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1750   \pgfcoordinate { \@@_env: - col - ##1 }
1751   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1752 }

```

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

```

1753 \@@_create_diag_nodes:

```

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

```

1754 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1755 \endpgfpicture

```

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

```

1756 \@@_create_blocks_nodes:
1757 \IfPackageLoadedTF { tikz }
1758 {
1759   \tikzset
1760   {
1761     every-picture / .style =
1762     { overlay , name-prefix = \@@_env: - }
1763   }
1764 }
1765 { }
1766 \cs_set_eq:NN \cellcolor \@@_cellcolor
1767 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1768 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1769 \cs_set_eq:NN \rowcolor \@@_rowcolor
1770 \cs_set_eq:NN \rowcolors \@@_rowcolors
1771 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1772 \cs_set_eq:NN \arraycolor \@@_arraycolor
1773 \cs_set_eq:NN \columncolor \@@_columncolor
1774 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1775 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1776 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1777 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1778 }

```

```

1779 \cs_new_protected:Npn \@@_exec_code_before:
1780 {
1781   \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`.

```

1782 \@@_add_to_colors_seq:nn { { nocolor } } { }
1783 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1784 \group_begin:

```

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

```

1785 \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`).

```
1786 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1787 { \@@_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`.

```
1788 \exp_last_unbraced:Nv \@@_CodeBefore_keys:
1789 \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.

```
1790 \@@_actually_color:
1791 \l_@@_code_before_tl
1792 \q_stop
1793 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1794 \group_end:
1795 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1796 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1797 }
```

```
1798 \keys_define:nn { NiceMatrix / CodeBefore }
1799 {
1800   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1801   create-cell-nodes .default:n = true ,
1802   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1803   sub-matrix .value_required:n = true ,
1804   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1805   delimiters / color .value_required:n = true ,
1806   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1807 }

1808 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1809 {
1810   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1811   \@@_CodeBefore:w
1812 }
```

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.

```
1813 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1814 {
1815   \bool_if:NT \g_@@_aux_found_bool
1816   {
1817     \@@_pre_code_before:
1818     #1
1819   }
1820 }
```

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.

```
1821 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1822 {
1823   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
```

```

1824 {
1825   \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1826   \pgfcoordinate { \@@_env: - row - ##1 - base }
1827   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1828   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1829   {
1830     \cs_if_exist:cT
1831     { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1832     {
1833       \pgfsys@getposition
1834       { \@@_env: - ##1 - #####1 - NW }
1835       \@@_node_position:
1836       \pgfsys@getposition
1837       { \@@_env: - ##1 - #####1 - SE }
1838       \@@_node_position_i:
1839       \@@_pgf_rect_node:nnn
1840       { \@@_env: - ##1 - #####1 }
1841       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1842       { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1843     }
1844   }
1845 }
1846 \int_step_inline:nn \c@iRow
1847 {
1848   \pgfnodealias
1849   { \@@_env: - ##1 - last }
1850   { \@@_env: - ##1 - \int_use:N \c@jCol }
1851 }
1852 \int_step_inline:nn \c@jCol
1853 {
1854   \pgfnodealias
1855   { \@@_env: - last - ##1 }
1856   { \@@_env: - \int_use:N \c@iRow - ##1 }
1857 }
1858 \@@_create_extra_nodes:
1859 }

1860 \cs_new_protected:Npn \@@_create_blocks_nodes:
1861 {
1862   \pgfpicture
1863   \pgf@relevantforpicturesizefalse
1864   \pgfrememberpicturepositiononpagetrue
1865   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1866   { \@@_create_one_block_node:nnnnn ##1 }
1867   \endpgfpicture
1868 }

```

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>

```

1869 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1870 {
1871   \tl_if_empty:nF { #5 }
1872   {
1873     \@@_qpoint:n { col - #2 }
1874     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1875     \@@_qpoint:n { #1 }
1876     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1877     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1878     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```

<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).

```

1879     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1880     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1881     \@@_pgf_rect_node:nnnnn
1882     { \@@_env: - #5 }
1883     { \dim_use:N \l_tmpa_dim }
1884     { \dim_use:N \l_tmpb_dim }
1885     { \dim_use:N \l_@@_tmpc_dim }
1886     { \dim_use:N \l_@@_tmpd_dim }
1887   }
1888 }

1889 \cs_new_protected:Npn \@@_patch_for_revtext:
1890 {
1891   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1892   \cs_set_eq:NN \insert@column \insert@column@array
1893   \cs_set_eq:NN \@classx \@classx@array
1894   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1895   \cs_set_eq:NN \@arraycr \@arraycr@array
1896   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1897   \cs_set_eq:NN \array \array@array
1898   \cs_set_eq:NN \@array \@array@array
1899   \cs_set_eq:NN \@tabular \@tabular@array
1900   \cs_set_eq:NN \mkpream \mkpream@array
1901   \cs_set_eq:NN \endarray \endarray@array
1902   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1903   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1904 }

```

## 11 The environment {NiceArrayWithDelims}

```

1905 \NewDocumentEnvironment { NiceArrayWithDelims }
1906 { m m O { } m ! O { } t \CodeBefore }
1907 {
1908   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1909   \@@_provide_pgfsyspdfmark:
1910   \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.

```

1911   \bgroup

1912   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1913   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1914   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1915   \int_gzero:N \g_@@_block_box_int
1916   \dim_zero:N \g_@@_width_last_col_dim
1917   \dim_zero:N \g_@@_width_first_col_dim
1918   \bool_gset_false:N \g_@@_row_of_col_done_bool
1919   \str_if_empty:NT \g_@@_name_env_str
1920     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1921   \bool_if:NTF \l_@@_tabular_bool
1922     \mode_leave_vertical:
1923     \@@_test_if_math_mode:
1924   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1925   \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.

```
1926 \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.

```
1927 \cs_if_exist:NT \tikz@library@external@loaded
1928 {
1929     \tikzexternaldisable
1930     \cs_if_exist:NT \ifstandalone
1931     { \tikzset { external / optimize = false } }
1932 }
```

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

```
1933 \int_gincr:N \g_@@_env_int
1934 \bool_if:NF \l_@@_block_auto_columns_width_bool
1935 { \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`).

```
1936 \seq_gclear:N \g_@@_blocks_seq
1937 \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`.

```
1938 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1939 \seq_gclear:N \g_@@_pos_of_xdots_seq
1940 \tl_gclear_new:N \g_@@_code_before_tl
1941 \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.

```
1942 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1943 {
1944     \bool_gset_true:N \g_@@_aux_found_bool
1945     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1946 }
1947 { \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.

```
1948 \tl_gclear:N \g_@@_aux_tl
1949 \tl_if_empty:NF \g_@@_code_before_tl
1950 {
1951     \bool_set_true:N \l_@@_code_before_bool
1952     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1953 }
1954 \tl_if_empty:NF \g_@@_pre_code_before_tl
1955 { \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`.

```
1956 \bool_if:NTF \g_@@_delims_bool
1957 { \keys_set:nn { NiceMatrix / pNiceArray } }
1958 { \keys_set:nn { NiceMatrix / NiceArray } }
1959 { #3 , #5 }
```

---

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

```
1960 \@@_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:.

```
1961 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1962 }
```

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

```
1963 {
1964   \bool_if:NTF \l_@@_light_syntax_bool
1965     { \use:c { end @@-light-syntax } }
1966     { \use:c { end @@-normal-syntax } }
1967   \c_math_toggle_token
1968   \skip_horizontal:N \l_@@_right_margin_dim
1969   \skip_horizontal:N \l_@@_extra_right_margin_dim
1970   \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.

```
1971 \bool_if:NT \l_@@_width_used_bool
1972 {
1973   \int_if_zero:nT \g_@@_total_X_weight_int
1974     { \@@_error_or_warning:n { width-without-X~columns } }
1975 }
```

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  $n$ .

```
1976 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1977 {
1978   \tl_gput_right:Nx \g_@@_aux_tl
1979   {
1980     \bool_set_true:N \l_@@_X_columns_aux_bool
1981     \dim_set:Nn \l_@@_X_columns_dim
1982     {
1983       \dim_compare:nNnTF
1984       {
1985         \dim_abs:n
1986         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1987       }
1988       <
1989       { 0.001 pt }
1990       { \dim_use:N \l_@@_X_columns_dim }
1991       {
1992         \dim_eval:n
1993         {
1994           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1995           / \int_use:N \g_@@_total_X_weight_int
1996           + \l_@@_X_columns_dim
1997         }
1998       }
1999     }
2000   }
2001 }
```

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

```
2002 \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```

2003 {
2004   \bool_if:NF \l_@@_last_row_without_value_bool
2005   {
2006     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2007     {
2008       \@@_error:n { Wrong-last-row }
2009       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2010     }
2011   }
2012 }

```

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>

```

2013   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2014   \bool_if:NTF \g_@@_last_col_found_bool
2015   { \int_gdecr:N \c@jCol }
2016   {
2017     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2018     { \@@_error:n { last-col-not-used } }
2019   }

```

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

```

2020   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2021   \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. 89).

```

2022   \int_if_zero:nT \l_@@_first_col_int
2023   { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2024   \bool_if:nTF { ! \g_@@_delims_bool }
2025   {
2026     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2027     \@@_use_arraybox_with_notes_c:
2028     {
2029       \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2030       \@@_use_arraybox_with_notes_b:
2031       \@@_use_arraybox_with_notes:
2032     }
2033   }

```

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).

```

2034   {
2035     \int_if_zero:nTF \l_@@_first_row_int
2036     {
2037       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2038       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2039     }
2040     { \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>

```

2041   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2042   {
2043     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2044     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2045   }

```

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

<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).



```

2046      { \dim_zero:N \l_tmpb_dim }
2047 \hbox_set:Nn \l_tmpa_box
2048 {
2049   \c_math_toggle_token
2050   \@@_color:o \l_@@_delimiters_color_tl
2051   \exp_after:wN \left \g_@@_left_delim_tl
2052   \vcenter
2053   {

```

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.

```

2054       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2055       \hbox
2056       {
2057         \bool_if:NTF \l_@@_tabular_bool
2058         { \skip_horizontal:N -\tabcolsep }
2059         { \skip_horizontal:N -\arraycolsep }
2060         \@@_use_arraybox_with_notes_c:
2061         \bool_if:NTF \l_@@_tabular_bool
2062         { \skip_horizontal:N -\tabcolsep }
2063         { \skip_horizontal:N -\arraycolsep }
2064       }

```

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

```

2065       \skip_vertical:N -\l_tmpb_dim
2066       \skip_vertical:N \arrayrulewidth
2067     }
2068     \exp_after:wN \right \g_@@_right_delim_tl
2069     \c_math_toggle_token
2070   }

```

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.

```

2071     \bool_if:NTF \l_@@_delimiters_max_width_bool
2072     {
2073       \@@_put_box_in_flow_bis:nn
2074       \g_@@_left_delim_tl
2075       \g_@@_right_delim_tl
2076     }
2077     \@@_put_box_in_flow:
2078   }

```

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. 90).

```

2079     \bool_if:NT \g_@@_last_col_found_bool
2080     { \skip_horizontal:N \g_@@_width_last_col_dim }
2081     \bool_if:NT \l_@@_preamble_bool
2082     {
2083       \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2084       { \@@_warning_gredirect_none:n { columns~not~used } }
2085     }
2086     \@@_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.

```

2087   \egroup

```

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

```

2088   \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2089   \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2090   \iow_now:Nx \@mainaux
2091   {
2092     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2093     { \exp_not:o \g_@@_aux_tl }

```

```

2094     }
2095     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2096     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2097 }

```

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.

```

2098 \cs_new_protected:Npn \@_transform_preamble:
2099 {
2100     \@_transform_preamble_i:
2101     \@_transform_preamble_ii:
2102 }
2103 \cs_new_protected:Npn \@_transform_preamble_i:
2104 {
2105     \int_gzero:N \c@jCol

```

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

```

2106     \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.

```

2107     \bool_gset_false:N \g_tmpb_bool

```

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

```

2108     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2109     \int_zero:N \l_tmpa_int
2110     \tl_gclear:N \g_@@_array_preamble_tl
2111     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2112     {
2113         \tl_gset:Nn \g_@@_array_preamble_tl
2114             { ! { \skip_horizontal:N \arrayrulewidth } }
2115     }
2116     {
2117         \clist_if_in:NnT \l_@@_vlines_clist 1
2118         {
2119             \tl_gset:Nn \g_@@_array_preamble_tl
2120                 { ! { \skip_horizontal:N \arrayrulewidth } }
2121         }
2122     }

```

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

```

2123     \exp_last_unbraced:NV \@_rec_preamble:n \g_@@_user_preamble_tl \stop
2124     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2125     \@_replace_columncolor:
2126 }

```

```

2127 \hook_gput_code:nnn { begindocument } { . }
2128 {
2129   \IfPackageLoadedTF { colortbl }
2130   {
2131     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2132     \cs_new_protected:Npn \@@_replace_columncolor:
2133     {
2134       \regex_replace_all:NnN
2135       \c_@@_columncolor_regex
2136       { \c { @@_columncolor_preamble } }
2137       \g_@@_array_preamble_tl
2138     }
2139   }
2140   {
2141     \cs_new_protected:Npn \@@_replace_columncolor:
2142     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2143   }
2144 }

2145 \cs_new_protected:Npn \@@_transform_preamble_ii:
2146 {

```

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

```

2147   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2148   {
2149     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2150     { \bool_gset_true:N \g_@@_delims_bool }
2151   }
2152   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2153   \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).

```

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

```

```

2180     }
2181   }
2182 }
2183 }

```

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`).

```

2184   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2185   {
2186     \tl_gput_right:Nn \g_@@_array_preamble_tl
2187     { > { \@@_error_too_much_cols: } 1 }
2188   }
2189 }

```

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.

```

2190 \cs_new_protected:Npn \@@_rec_preamble:n #1
2191 {

```

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>

```

2192   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2193   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2194   {

```

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

```

2195     \cs_if_exist:cTF { NC @ find @ #1 }
2196     {
2197       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2198       \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2199     }
2200     {
2201       \tl_if_eq:nnT { #1 } { S }
2202       { \@@_fatal:n { unknown~column~type~S } }
2203       { \@@_fatal:nn { unknown~column~type } { #1 } }
2204     }
2205   }
2206 }

```

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

```

2207 \cs_new:Npn \@@_c #1
2208 {
2209   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2210   \tl_gclear:N \g_@@_pre_cell_tl
2211   \tl_gput_right:Nn \g_@@_array_preamble_tl
2212   { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2213   \int_gincr:N \c@jCol
2214   \@@_rec_preamble_after_col:n
2215 }

2216 \cs_new:Npn \@@_l #1
2217 {
2218   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2219   \tl_gclear:N \g_@@_pre_cell_tl
2220   \tl_gput_right:Nn \g_@@_array_preamble_tl

```

---

<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`.

```

2221     {
2222     > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2223     l
2224     < \@@_cell_end:
2225     }
2226     \int_gincr:N \c@jCol
2227     \@@_rec_preamble_after_col:n
2228   }
2229   \cs_new:Npn \@@_r #1
2230   {
2231     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2232     \tl_gclear:N \g_@@_pre_cell_tl
2233     \tl_gput_right:Nn \g_@@_array_preamble_tl
2234     {
2235       > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2236       r
2237       < \@@_cell_end:
2238     }
2239     \int_gincr:N \c@jCol
2240     \@@_rec_preamble_after_col:n
2241   }

```

For ! and @

```

2242   \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2243   {
2244     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2245     \@@_rec_preamble:n
2246   }
2247   \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2248   \cs_new:cpn { @@ _ | } #1
2249   {

```

\l\_tmpa\_int is the number of successive occurrences of |

```

2250     \int_incr:N \l_tmpa_int
2251     \@@_make_preamble_i_i:n
2252   }
2253   \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2254   {
2255     \str_if_eq:nnTF { #1 } |
2256     { \use:c { @@ _ | } | }
2257     { \@@_make_preamble_i_ii:nn { } #1 }
2258   }
2259   \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2260   {
2261     \str_if_eq:nnTF { #2 } [
2262     { \@@_make_preamble_i_iii:nn { #1 } [ ] }
2263     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2264   ]
2265   \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 [ #2 ]
2266   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2267   \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2268   {
2269     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2270     \tl_gput_right:Nx \g_@@_array_preamble_tl
2271     {

```

Here, the command \dim\_eval:n is mandatory.

```

2272     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2273   }
2274   \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

2275 {
2276   \@@_vline:n
2277   {
2278     position = \int_eval:n { \c@jCol + 1 } ,
2279     multiplicity = \int_use:N \l_tmpa_int ,
2280     total-width = \dim_use:N \l_@@_rule_width_dim ,
2281     #2
2282   }

```

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.

```

2283   }
2284   \int_zero:N \l_tmpa_int
2285   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2286   \@@_rec_preamble:n #1
2287 }

```

```

2288 \cs_new:cpn { @@ _ > } #1 #2
2289 {
2290   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2291   \@@_rec_preamble:n
2292 }
2293 \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.

```

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

```

```

2330     t .meta:n = p ,
2331     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2332     m .value_forbidden:n = true ,
2333     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2334     b .value_forbidden:n = true ,
2335 }

```

For p but also b and m.

```

2336 \cs_new:Npn \@@_p #1
2337 {
2338   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2339   \@@_make_preamble_ii_i:n
2340 }
2341 \cs_set_eq:NN \@@_b \@@_p
2342 \cs_set_eq:NN \@@_m \@@_p
2343 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2344 {
2345   \str_if_eq:nnTF { #1 } { [ ]
2346     { \@@_make_preamble_ii_ii:w [ ]
2347       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2348     }
2349   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2350     { \@@_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.

```

2351 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2352 {

```

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).

```

2353   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2354   \@@_keys_p_column:n { #1 }
2355   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2356 }
2357 \cs_new_protected:Npn \@@_keys_p_column:n #1
2358 { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }

```

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

```

2359 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2360 {
2361   \use:e
2362   {
2363     \@@_make_preamble_ii_v:nnnnnnnn
2364     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2365     { \dim_eval:n { #1 } }
2366   }

```

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.

```

2367   \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2368   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2369   {
2370     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2371     { \str_lowercase:V \l_@@_hpos_col_str }
2372   }
2373   \str_case:on \l_@@_hpos_col_str

```

```

2374         {
2375             c { \exp_not:N \centering }
2376             l { \exp_not:N \raggedright }
2377             r { \exp_not:N \raggedleft }
2378             C { \exp_not:N \Centering }
2379             L { \exp_not:N \RaggedRight }
2380             R { \exp_not:N \RaggedLeft }
2381         }
2382     #3
2383 }
2384 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2385 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2386 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2387 { #2 }
2388 {
2389     \str_case:onF \l_@@_hpos_col_str
2390     {
2391         { j } { c }
2392         { si } { c }
2393     }

```

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

```

2394     { \str_lowercase:V \l_@@_hpos_col_str }
2395 }
2396 }

```

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

```

2397     \int_gincr:N \c@jCol
2398     \@@_rec_preamble_after_col:n
2399 }

```

**#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*.

```

2400 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2401 {
2402     \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2403     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2404     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2405     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2406     \tl_gclear:N \g_@@_pre_cell_tl
2407     \tl_gput_right:Nn \g_@@_array_preamble_tl
2408     {
2409         > {

```

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.

```

2410     \dim_set:Nn \l_@@_col_width_dim { #2 }
2411     \bool_if:NT \c_@@_tagging_array_bool
2412     { \tag_struct_begin:n { tag = Div } }
2413     \@@_cell_begin:w

```



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

```
2414 \use:c { #7 } [ #1 ] { #2 }
```

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

```
2415 \everypar
2416 {
2417   \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2418   \everypar { }
2419 }
2420 \bool_if:NT \c_@@_tagging_array_bool \tagpdfparaOn
```

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

```
2421 #3
```

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

```
2422 \g_@@_row_style_tl
2423 \arraybackslash
2424 #5
2425 }
2426 #8
2427 < {
2428 #6
```

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

```
2429 \@finalstrut \@arstrutbox
2430 \use:c { end #7 }
```

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

```
2431 #4
2432 \@_cell_end:
2433 \bool_if:NT \c_@@_tagging_array_bool \tag_struct_end:
2434 }
2435 }
2436 }
```

```
2437 \str_new:N \c_@@_ignorespaces_str
2438 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2439 \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...

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

```

2458 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2459 {
2460   \peek_meaning:NT \_siunitx_table_skip:n
2461   {
2462     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2463     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2464   }
2465 }

```

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

```

2466 \cs_new_protected:Npn \@@_center_cell_box:
2467 {

```

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

```

2468   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2469   {
2470     \int_compare:nNnT
2471       { \box_ht:N \l_@@_cell_box }
2472     >

```

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).

```

2473       { \box_ht:N \strutbox }
2474     {
2475       \hbox_set:Nn \l_@@_cell_box
2476       {
2477         \box_move_down:nn
2478         {
2479           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2480             + \baselineskip ) / 2
2481         }
2482         { \box_use:N \l_@@_cell_box }
2483       }
2484     }
2485   }
2486 }

```

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

```

2487 \cs_new:Npn \@@_V #1 #2
2488 {
2489   \str_if_eq:nnTF { #2 } { [ ] }
2490   { \@@_make_preamble_V_i:w [ ] }
2491   { \@@_make_preamble_V_i:w [ ] { #2 } }
2492 }
2493 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2494 { \@@_make_preamble_V_ii:nn { #1 } }
2495 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2496 {
2497   \str_set:Nn \l_@@_vpos_col_str { p }
2498   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2499   \@@_keys_p_column:n { #1 }
2500   \IfPackageLoadedTF { varwidth }
2501   { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2502   {
2503     \@@_error_or_warning:n { varwidth-not-loaded }
2504     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2505   }
2506 }

```

For w and W

```

2507 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2508 \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.

2509 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2510 {
2511   \str_if_eq:nnTF { #3 } { s }
2512   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2513   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2514 }

```

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.

```

2515 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2516 {
2517   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2518   \tl_gclear:N \g_@@_pre_cell_tl
2519   \tl_gput_right:Nn \g_@@_array_preamble_tl
2520   {
2521     > {
2522       \dim_set:Nn \l_@@_col_width_dim { #2 }
2523       \@@_cell_begin:w
2524       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2525     }
2526     c
2527     < {
2528       \@@_cell_end_for_w_s:
2529       #1
2530       \@@_adjust_size_box:
2531       \box_use_drop:N \l_@@_cell_box
2532     }
2533   }
2534   \int_gincr:N \c@jCol
2535   \@@_rec_preamble_after_col:n
2536 }

```

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

```

2537 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2538 {
2539   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2540   \tl_gclear:N \g_@@_pre_cell_tl
2541   \tl_gput_right:Nn \g_@@_array_preamble_tl
2542   {
2543     > {

```

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.

```

2544       \dim_set:Nn \l_@@_col_width_dim { #4 }
2545       \hbox_set:Nw \l_@@_cell_box
2546       \@@_cell_begin:w
2547       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2548     }
2549     c
2550     < {
2551       \@@_cell_end:
2552       \hbox_set_end:
2553       #1

```

```

2554         \@@_adjust_size_box:
2555         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2556     }
2557 }

```

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

```

2558     \int_gincr:N \c@jCol
2559     \@@_rec_preamble_after_col:n
2560 }

```

```

2561 \cs_new_protected:Npn \@@_special_W:
2562 {
2563     \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2564     { \@@_warning:n { W~warning } }
2565 }

```

For S (of siunitx).

```

2566 \cs_new:Npn \@@_S #1 #2
2567 {
2568     \str_if_eq:nnTF { #2 } { [ ] }
2569     { \@@_make_preamble_S:w [ ] }
2570     { \@@_make_preamble_S:w [ ] { #2 } }
2571 }
2572 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2573 { \@@_make_preamble_S:i:n { #1 } }
2574 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2575 {
2576     \IfPackageLoadedTF { siunitx }
2577     {
2578         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2579         \tl_gclear:N \g_@@_pre_cell_tl
2580         \tl_gput_right:Nn \g_@@_array_preamble_tl
2581         {
2582             > {
2583                 \@@_cell_begin:w
2584                 \keys_set:nn { siunitx } { #1 }
2585                 \siunitx_cell_begin:w
2586             }
2587             c
2588             < { \siunitx_cell_end: \@@_cell_end: }
2589         }

```

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

```

2590     \int_gincr:N \c@jCol
2591     \@@_rec_preamble_after_col:n
2592 }
2593 { \@@_fatal:n { siunitx~not~loaded } }
2594 }

```

For (, [ and \{.

```

2595 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2596 {
2597     \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.

```

2598     \int_if_zero:nTF \c@jCol
2599     {
2600         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2601         {

```

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

```

2602         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2603         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2604         \@@_rec_preamble:n #2
2605     }
2606     {
2607         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2608         \@@_make_preamble_iv:nn { #1 } { #2 }
2609     }
2610 }
2611 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2612 }
2613 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( ) }
2614 \cs_set_eq:cc { @@ _ \token_to_str:N \{ \} { @@ _ \token_to_str:N ( ) }
2615 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2616 {
2617     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2618     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2619     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2620     {
2621         \@@_error:nn { delimiter~after~opening } { #2 }
2622         \@@_rec_preamble:n
2623     }
2624     { \@@_rec_preamble:n #2 }
2625 }

```

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

```

2626 \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}`).

```

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

```

```

2655 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2656 {
2657   \tl_if_eq:nnTF { \stop } { #3 }
2658   {
2659     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2660     {
2661       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2662       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2663       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2664       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2665     }
2666     {
2667       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2668       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2669       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2670       \@@_error:nn { double~closing~delimiter } { #2 }
2671     }
2672   }
2673   {
2674     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2675     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2676     \@@_error:nn { double~closing~delimiter } { #2 }
2677     \@@_rec_preamble:n #3
2678   }
2679 }

2680 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2681 { \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 @{...}.

```

2682 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2683 {
2684   \str_if_eq:nnTF { #1 } { < }
2685   \@@_rec_preamble_after_col_i:n
2686   {
2687     \str_if_eq:nnTF { #1 } { @ }
2688     \@@_rec_preamble_after_col_ii:n
2689     {
2690       \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2691       {
2692         \tl_gput_right:Nn \g_@@_array_preamble_tl
2693         { ! { \skip_horizontal:N \arrayrulewidth } }
2694       }
2695       {
2696         \exp_args:NNe
2697         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2698         {
2699           \tl_gput_right:Nn \g_@@_array_preamble_tl
2700           { ! { \skip_horizontal:N \arrayrulewidth } }
2701         }
2702       }
2703       \@@_rec_preamble:n { #1 }
2704     }
2705   }
2706 }

2707 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2708 {
2709   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2710   \@@_rec_preamble_after_col:n
2711 }

```

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.

```

2712 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2713 {
2714   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2715   {
2716     \tl_gput_right:Nn \g_@@_array_preamble_tl
2717     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2718   }
2719   {
2720     \exp_args:NNe
2721     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2722     {
2723       \tl_gput_right:Nn \g_@@_array_preamble_tl
2724       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2725     }
2726     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2727   }
2728   \@@_rec_preamble:n
2729 }

2730 \cs_new:cpn { @@ _ * } #1 #2 #3
2731 {
2732   \tl_clear:N \l_tmpa_tl
2733   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2734   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2735 }

```

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.

```

2736 \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.

```

2737 \cs_new:Npn \@@_X #1 #2
2738 {
2739   \str_if_eq:nnTF { #2 } { [ ]
2740     { \@@_make_preamble_X:w [ ] }
2741     { \@@_make_preamble_X:w [ ] #2 }
2742   }

2743   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2744   { \@@_make_preamble_X:i:n { #1 } }

```

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

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

```

2745 \keys_define:nn { nicematrix / X-column }
2746 { 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.

```

2747 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2748 {

```

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).

```

2749   \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).

```

2750   \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`.

```

2751 \int_zero_new:N \l_@@_weight_int
2752 \int_set_eq:NN \l_@@_weight_int \c_one_int
2753 \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2754 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2755 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2756 {
2757   \@@_error_or_warning:n { negative-weight }
2758   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2759 }
2760 \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).

```

2761 \bool_if:NTF \l_@@_X_columns_aux_bool
2762 {
2763   \exp_args:Nne
2764   \@@_make_preamble_ii_iv:nnn
2765   { \l_@@_weight_int \l_@@_X_columns_dim }
2766   { minipage }
2767   { \@@_no_update_width: }
2768 }
2769 {
2770   \tl_gput_right:Nn \g_@@_array_preamble_tl
2771   {
2772     > {
2773       \@@_cell_begin:w
2774       \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`.

```

2775 \NotEmpty

```

The following code will nullify the box of the cell.

```

2776 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2777 { \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`.

```

2778 \begin { minipage } { 5 cm } \arraybackslash
2779 }
2780 c
2781 < {
2782   \end { minipage }
2783   \@@_cell_end:
2784 }
2785 }
2786 \int_gincr:N \c@jCol
2787 \@@_rec_preamble_after_col:n
2788 }
2789 }

```

```

2790 \cs_new_protected:Npn \@@_no_update_width:
2791 {
2792   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2793   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2794 }

```



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

```

2795 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2796 {
2797   \seq_gput_right:Nx \g_@@_cols_vlism_seq
2798   { \int_eval:n { \c@jCol + 1 } }
2799   \tl_gput_right:Nx \g_@@_array_preamble_tl
2800   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2801   \@@_rec_preamble:n
2802 }

```

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.

```

2803 \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).

```

2804 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2805 { \@@_fatal:n { Preamble-forgotten } }
2806 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2807 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2808 \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).

```

2809 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2810 {

```

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

```

2811   \multispan { #1 }
2812   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2813   \begingroup
2814   \bool_if:NT \c_@@_tagging_array_bool
2815   { \tbl_update_multicolumn_cell_data:n { #1 } }
2816   \cs_set:Npn \@addamp
2817   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2818   \tl_gclear:N \g_@@_preamble_tl
2819   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2820   \exp_args:No \@mkpream \g_@@_preamble_tl
2821   \@addtopreamble \empty
2822   \endgroup
2823   \bool_if:NT \c_@@_tagging_array_bool
2824   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2825   \int_compare:nNnT { #1 } > \c_one_int
2826   {
2827     \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2828     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2829     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }

```

```

2830 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2831 {
2832   {
2833     \int_if_zero:nTF \c@jCol
2834       { \int_eval:n { \c@iRow + 1 } }
2835       { \int_use:N \c@iRow }
2836   }
2837   { \int_eval:n { \c@jCol + 1 } }
2838   {
2839     \int_if_zero:nTF \c@jCol
2840       { \int_eval:n { \c@iRow + 1 } }
2841       { \int_use:N \c@iRow }
2842   }
2843   { \int_eval:n { \c@jCol + #1 } }
2844   { } % for the name of the block
2845 }
2846 }

```

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

```

2847 \cs_set:Npn \@sharp { #3 }
2848 \@arstrut
2849 \@preamble
2850 \null

```

We add some lines.

```

2851 \int_gadd:Nn \c@jCol { #1 - 1 }
2852 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2853   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2854 \ignorespaces
2855 }

```

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`.

```

2856 \cs_new_protected:Npn \@_make_m_preamble:n #1
2857 {
2858   \str_case:nnF { #1 }
2859   {
2860     c { \@_make_m_preamble_i:n #1 }
2861     l { \@_make_m_preamble_i:n #1 }
2862     r { \@_make_m_preamble_i:n #1 }
2863     > { \@_make_m_preamble_ii:nn #1 }
2864     ! { \@_make_m_preamble_ii:nn #1 }
2865     @ { \@_make_m_preamble_ii:nn #1 }
2866     | { \@_make_m_preamble_iii:n #1 }
2867     p { \@_make_m_preamble_iv:nnn t #1 }
2868     m { \@_make_m_preamble_iv:nnn c #1 }
2869     b { \@_make_m_preamble_iv:nnn b #1 }
2870     w { \@_make_m_preamble_v:nnnn { } #1 }
2871     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2872     \q_stop { }
2873   }
2874   {
2875     \cs_if_exist:cTF { NC @ find @ #1 }
2876     {
2877       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2878       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2879     }
2880     {
2881       \tl_if_eq:nnT { #1 } { S }
2882       { \@_fatal:n { unknown~column~type~S } }
2883       { \@_fatal:nn { unknown~column~type } { #1 } }
2884     }
2885   }

```

2886 }

For c, l and r

```
2887 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2888 {
2889   \tl_gput_right:Nn \g_@@_preamble_tl
2890   {
2891     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2892     #1
2893     < \@@_cell_end:
2894   }
```

We test for the presence of a <.

```
2895   \@@_make_m_preamble_x:n
2896 }
```

For >, ! and @

```
2897 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2898 {
2899   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2900   \@@_make_m_preamble:n
2901 }
```

For |

```
2902 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2903 {
2904   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2905   \@@_make_m_preamble:n
2906 }
```

For p, m and b

```
2907 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2908 {
2909   \tl_gput_right:Nn \g_@@_preamble_tl
2910   {
2911     > {
2912       \@@_cell_begin:w
2913       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2914       \mode_leave_vertical:
2915       \arraybackslash
2916       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2917     }
2918     c
2919     < {
2920       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2921       \end { minipage }
2922       \@@_cell_end:
2923     }
2924   }
```

We test for the presence of a <.

```
2925   \@@_make_m_preamble_x:n
2926 }
```

For w and W

```
2927 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2928 {
2929   \tl_gput_right:Nn \g_@@_preamble_tl
2930   {
2931     > {
2932       \dim_set:Nn \l_@@_col_width_dim { #4 }
2933       \hbox_set:Nw \l_@@_cell_box
2934       \@@_cell_begin:w
2935       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
```

```

2936     }
2937     c
2938     < {
2939         \@@_cell_end:
2940         \hbox_set_end:
2941         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2942         #1
2943         \@@_adjust_size_box:
2944         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2945     }
2946 }

```

We test for the presence of a <.

```

2947     \@@_make_m_preamble_x:n
2948 }

```

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

```

2949 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2950 {
2951     \str_if_eq:nnTF { #1 } { < }
2952     \@@_make_m_preamble_ix:n
2953     { \@@_make_m_preamble:n { #1 } }
2954 }
2955 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2956 {
2957     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2958     \@@_make_m_preamble_x:n
2959 }

```

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`).

```

2960 \cs_new_protected:Npn \@@_put_box_in_flow:
2961 {
2962     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2963     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2964     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2965     { \box_use_drop:N \l_tmpa_box }
2966     \@@_put_box_in_flow_i:
2967 }

```

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).

```

2968 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2969 {
2970     \pgfpicture
2971     \@@_qpoint:n { row - 1 }
2972     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2973     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2974     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2975     \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).

```

2976     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2977     {
2978         \int_set:Nn \l_tmpa_int
2979         {
2980             \str_range:Nnn
2981             \l_@@_baseline_tl
2982             6

```

```

2983         { \tl_count:o \l_@@_baseline_tl }
2984     }
2985     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2986 }
2987 {
2988     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
2989     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2990     {
2991         \tl_if_eq:NnTF \l_@@_baseline_tl { b }
2992         { \int_set_eq:NN \l_tmpa_int \c_iRow }
2993         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2994     }
2995     \bool_lazy_or:nnT
2996     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2997     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2998     {
2999         \@@_error:n { bad-value-for-baseline }
3000         \int_set_eq:NN \l_tmpa_int \c_one_int
3001     }
3002     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3003     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3004 }
3005 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3006 \endpgfpicture
3007 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3008 \box_use_drop:N \l_tmpa_box
3009 }

```

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).

```

3010 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3011 {

```

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.

```

3012     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3013     {
3014         \int_compare:nNnT \c_jCol > \c_one_int
3015         {
3016             \box_set_wd:Nn \l_@@_the_array_box
3017             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3018         }
3019     }

```

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

```

3020     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3021     \bool_if:NT \l_@@_caption_above_bool
3022     {
3023         \tl_if_empty:NF \l_@@_caption_tl
3024         {
3025             \bool_set_false:N \g_@@_caption_finished_bool
3026             \int_gzero:N \c@tabularnote
3027             \@@_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).

```

3028         \int_compare:nNtT \g_@@_notes_caption_int > \c_zero_int
3029         {
3030             \tl_gput_right:Nx \g_@@_aux_tl
3031             {
3032                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3033                 { \int_use:N \g_@@_notes_caption_int }
3034             }
3035             \int_gzero:N \g_@@_notes_caption_int
3036         }
3037     }
3038 }

```

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

```

3039     \hbox
3040     {
3041         \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.

```

3042         \@@_create_extra_nodes:
3043         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3044     }

```

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

```

3045     \bool_lazy_any:nT
3046     {
3047         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3048         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3049         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3050     }
3051     \@@_insert_tabularnotes:
3052     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3053     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3054     \end { minipage }
3055 }

```

```

3056 \cs_new_protected:Npn \@@_insert_caption:
3057 {
3058     \tl_if_empty:NF \l_@@_caption_tl
3059     {
3060         \cs_if_exist:NTF \@capytype
3061         { \@@_insert_caption_i: }
3062         { \@@_error:n { caption-outside-float } }
3063     }
3064 }

```

```

3065 \cs_new_protected:Npn \@@_insert_caption_i:
3066 {
3067     \group_begin:

```

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

```

3068     \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.

```

3069     \IfPackageLoadedTF { floatrow }

```

```

3070     { \cs_set_eq:NN \@makecaption \FR@makecaption }
3071     { }
3072     \tl_if_empty:NTF \l_@@_short_caption_tl
3073     { \caption }
3074     { \caption [ \l_@@_short_caption_tl ] }
3075     { \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.

```

3076     \bool_if:NF \g_@@_caption_finished_bool
3077     {
3078         \bool_gset_true:N \g_@@_caption_finished_bool
3079         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3080         \int_gzero:N \c@tabularnote
3081     }
3082     \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3083     \group_end:
3084 }

3085 \cs_new_protected:Npn \@_tabularnote_error:n #1
3086 {
3087     \@_error_or_warning:n { tabularnote~below~the~tabular }
3088     \@_gredirect_none:n { tabularnote~below~the~tabular }
3089 }

3090 \cs_new_protected:Npn \@_insert_tabularnotes:
3091 {
3092     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3093     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3094     \skip_vertical:N 0.65ex

```

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

```

3095     \group_begin:
3096     \l_@@_notes_code_before_tl
3097     \tl_if_empty:NF \g_@@_tabularnote_tl
3098     {
3099         \g_@@_tabularnote_tl \par
3100         \tl_gclear:N \g_@@_tabularnote_tl
3101     }

```

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.

```

3102     \int_compare:nNnT \c@tabularnote > \c_zero_int
3103     {
3104         \bool_if:NTF \l_@@_notes_para_bool
3105         {
3106             \begin { tabularnotes* }
3107             \seq_map_inline:Nn \g_@@_notes_seq
3108             { \@_one_tabularnote:nn ##1 }
3109             \strut
3110             \end { tabularnotes* }

```

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

```

3111         \par
3112     }
3113     {
3114         \tabularnotes
3115         \seq_map_inline:Nn \g_@@_notes_seq
3116         { \@_one_tabularnote:nn ##1 }
3117         \strut
3118         \endtabularnotes
3119     }

```

```

3120     }
3121     \unskip
3122     \group_end:
3123     \bool_if:NT \l_@@_notes_bottomrule_bool
3124     {
3125         \IfPackageLoadedTF { booktabs }
3126         {

```

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

```

3127         \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.

```

```

3128         { \CT@arc@ \hrule height \heavyrulewidth }
3129     }
3130     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3131 }
3132 \l_@@_notes_code_after_tl
3133 \seq_gclear:N \g_@@_notes_seq
3134 \seq_gclear:N \g_@@_notes_in_caption_seq
3135 \int_gzero:N \c@tabularnote
3136 }

```

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

```

3137 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3138 {
3139     \tl_if_novalue:nTF { #1 }
3140     { \item }
3141     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3142 }

```

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`.

```

3143 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3144 {
3145     \pgfpicture
3146     \@@_qpoint:n { row - 1 }
3147     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3148     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3149     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3150     \endpgfpicture
3151     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3152     \int_if_zero:nT \l_@@_first_row_int
3153     {
3154         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3155         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3156     }
3157     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3158 }

```

Now, the general case.

```

3159 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3160 {

```

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

```

3161     \tl_if_eq:NnT \l_@@_baseline_tl { t }
3162     { \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`.

```

3163 \pgfpicture
3164 \@@_qpoint:n { row - 1 }
3165 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3166 \str_if_in:NnTF \l_@@_baseline_tl { line- }
3167 {
3168   \int_set:Nn \l_tmpa_int
3169   {
3170     \str_range:Nnn
3171       \l_@@_baseline_tl
3172       6
3173     { \tl_count:o \l_@@_baseline_tl }
3174   }
3175   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3176 }
3177 {
3178   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3179   \bool_lazy_or:nnT
3180     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3181     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3182   {
3183     \@@_error:n { bad~value~for~baseline }
3184     \int_set:Nn \l_tmpa_int 1
3185   }
3186   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3187 }
3188 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3189 \endpgfpicture
3190 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3191 \int_if_zero:nT \l_@@_first_row_int
3192 {
3193   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3194   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3195 }
3196 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3197 }

```

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.

```

3198 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3199 {

```

We will compute the real width of both delimiters used.

```

3200   \dim_zero_new:N \l_@@_real_left_delim_dim
3201   \dim_zero_new:N \l_@@_real_right_delim_dim
3202   \hbox_set:Nn \l_tmpb_box
3203   {
3204     \c_math_toggle_token
3205     \left #1
3206     \vcenter
3207     {
3208       \vbox_to_ht:nn
3209         { \box_ht_plus_dp:N \l_tmpa_box }
3210         { }
3211     }
3212     \right .
3213     \c_math_toggle_token
3214   }
3215   \dim_set:Nn \l_@@_real_left_delim_dim
3216   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3217   \hbox_set:Nn \l_tmpb_box

```

```

3218 {
3219   \c_math_toggle_token
3220   \left .
3221   \vbox_to_ht:nn
3222     { \box_ht_plus_dp:N \l_tmpa_box }
3223     { }
3224   \right #2
3225   \c_math_toggle_token
3226 }
3227 \dim_set:Nn \l_@@_real_right_delim_dim
3228 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3229 \skip_horizontal:N \l_@@_left_delim_dim
3230 \skip_horizontal:N -\l_@@_real_left_delim_dim
3231 \@@_put_box_in_flow:
3232 \skip_horizontal:N \l_@@_right_delim_dim
3233 \skip_horizontal:N -\l_@@_real_right_delim_dim
3234 }

```

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).

```

3235 \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).

```

3236 {
3237   \peek_remove_spaces:n
3238   {
3239     \peek_meaning:NTF \end
3240     \@@_analyze_end:Nn
3241     {
3242       \@@_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`).

```

3243       \exp_args:No \@@_array: \g_@@_array_preamble_tl
3244     }
3245   }
3246 }
3247 {
3248   \@@_create_col_nodes:
3249   \endarray
3250 }

```

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

```

3251 \NewDocumentEnvironment { @@-light-syntax } { b }
3252 {

```

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.

```

3253 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3254 \tl_map_inline:nn { #1 }
3255 {
3256   \str_if_eq:nnT { ##1 } { & }
3257   { \@@_fatal:n { ampersand-in-light-syntax } }

```

```

3258     \str_if_eq:nnT { ##1 } { \\\ }
3259     { \@@_fatal:n { double-backslash-in-light-syntax } }
3260 }

```

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`.

```

3261 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3262 }

```

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.

```

3263 {
3264   \@@_create_col_nodes:
3265   \endarray
3266 }

3267 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3268 {
3269   \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).

```

3270 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3271 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3272 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3273   \seq_set_split:Nee
3274   \seq_set_split:NVn
3275   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3276 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3277 \tl_if_empty:NF \l_tmpa_tl
3278 { \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.

```

3279 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3280 { \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`.

```

3281 \tl_build_begin:N \l_@@_new_body_tl
3282 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3283 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3284 \@@_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).

```

3285 \seq_map_inline:Nn \l_@@_rows_seq
3286 {
3287   \tl_build_put_right:Nn \l_@@_new_body_tl { \\\ }
3288   \@@_line_with_light_syntax:n { ##1 }
3289 }
3290 \tl_build_end:N \l_@@_new_body_tl

```

```

3291 \int_compare:nNtT \l_@@_last_col_int = { -1 }
3292 {
3293   \int_set:Nn \l_@@_last_col_int
3294     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3295 }

```

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.

```

3296 \@@_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`).

```

3297 \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3298 }
3299 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3300 {
3301   \seq_clear_new:N \l_@@_cells_seq
3302   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3303   \int_set:Nn \l_@@_nb_cols_int
3304     {
3305       \int_max:nn
3306         \l_@@_nb_cols_int
3307         { \seq_count:N \l_@@_cells_seq }
3308     }
3309   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3310   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3311   \seq_map_inline:Nn \l_@@_cells_seq
3312     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3313 }
3314 \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`.

```

3315 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3316 {
3317   \str_if_eq:onT \g_@@_name_env_str { #2 }
3318     { \@@_fatal:n { empty-environment } }

```

We repute in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```

3319   \end { #2 }
3320 }

```

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).

```

3321 \cs_new:Npn \@@_create_col_nodes:
3322 {
3323   \crr
3324   \int_if_zero:nT \l_@@_first_col_int
3325     {
3326       \omit
3327       \hbox_overlap_left:n
3328         {
3329           \bool_if:NT \l_@@_code_before_bool
3330             { \pgfsys@markposition { \@@_env: - col - 0 } }
3331           \pgfpicture
3332             \pgfrememberpicturepositiononpagetrue
3333             \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3334             \str_if_empty:NF \l_@@_name_str

```

```

3335         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3336     \endpgfpicture
3337     \skip_horizontal:N 2\col@sep
3338     \skip_horizontal:N \g_@@_width_first_col_dim
3339 }
3340 &
3341 }
3342 \omit

```

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

```

3343     \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`).

```

3344     \int_if_zero:nTF \l_@@_first_col_int
3345     {
3346         \bool_if:NT \l_@@_code_before_bool
3347         {
3348             \hbox
3349             {
3350                 \skip_horizontal:N -0.5\arrayrulewidth
3351                 \pgfsys@markposition { \@@_env: - col - 1 }
3352                 \skip_horizontal:N 0.5\arrayrulewidth
3353             }
3354         }
3355         \pgfpicture
3356         \pgfrememberpicturepositiononpagetrue
3357         \pgfcoordinate { \@@_env: - col - 1 }
3358         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3359         \str_if_empty:NF \l_@@_name_str
3360         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3361         \endpgfpicture
3362     }
3363     {
3364         \bool_if:NT \l_@@_code_before_bool
3365         {
3366             \hbox
3367             {
3368                 \skip_horizontal:N 0.5\arrayrulewidth
3369                 \pgfsys@markposition { \@@_env: - col - 1 }
3370                 \skip_horizontal:N -0.5\arrayrulewidth
3371             }
3372         }
3373         \pgfpicture
3374         \pgfrememberpicturepositiononpagetrue
3375         \pgfcoordinate { \@@_env: - col - 1 }
3376         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3377         \str_if_empty:NF \l_@@_name_str
3378         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3379         \endpgfpicture
3380     }

```

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.

```

3381     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3382     \bool_if:NF \l_@@_auto_columns_width_bool
3383     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3384     {
3385         \bool_lazy_and:nnTF
3386         \l_@@_auto_columns_width_bool

```

```

3387         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3388         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3389         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3390     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3391 }
3392 \skip_horizontal:N \g_tmpa_skip
3393 \hbox
3394 {
3395     \bool_if:NT \l_@@_code_before_bool
3396     {
3397         \hbox
3398         {
3399             \skip_horizontal:N -0.5\arrayrulewidth
3400             \pgfsys@markposition { \@@_env: - col - 2 }
3401             \skip_horizontal:N 0.5\arrayrulewidth
3402         }
3403     }
3404     \pgfpicture
3405     \pgfrememberpicturepositiononpagetrue
3406     \pgfcoordinate { \@@_env: - col - 2 }
3407     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3408     \str_if_empty:NF \l_@@_name_str
3409     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3410     \endpgfpicture
3411 }

```

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.

```

3412 \int_gset_eq:NN \g_tmpa_int \c_one_int
3413 \bool_if:NTF \g_@@_last_col_found_bool
3414 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3415 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3416 {
3417     &
3418     \omit
3419     \int_gincr:N \g_tmpa_int

```

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

```

3420     \skip_horizontal:N \g_tmpa_skip
3421     \bool_if:NT \l_@@_code_before_bool
3422     {
3423         \hbox
3424         {
3425             \skip_horizontal:N -0.5\arrayrulewidth
3426             \pgfsys@markposition
3427             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3428             \skip_horizontal:N 0.5\arrayrulewidth
3429         }
3430     }

```

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

```

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

```

```

3443      &
3444      \omit

```

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

```

3445      \int_if_zero:nT \g_@@_col_total_int
3446      { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3447      \skip_horizontal:N \g_tmpa_skip
3448      \int_gincr:N \g_tmpa_int
3449      \bool_lazy_any:nF % modified 2023/12/13
3450      {
3451          \g_@@_delims_bool
3452          \l_@@_tabular_bool
3453          { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3454          \l_@@_exterior_arraycolsep_bool
3455          \l_@@_bar_at_end_of_pream_bool
3456      }
3457      { \skip_horizontal:N -\col@sep }
3458      \bool_if:NT \l_@@_code_before_bool
3459      {
3460          \hbox
3461          {
3462              \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.

```

3463          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3464          { \skip_horizontal:N -\arraycolsep }
3465          \pgfsys@markposition
3466          { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3467          \skip_horizontal:N 0.5\arrayrulewidth
3468          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3469          { \skip_horizontal:N \arraycolsep }
3470      }
3471  }
3472  \pgfpicture
3473  \pgfrememberpicturepositiononpagetrue
3474  \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3475  {
3476      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3477      {
3478          \pgfpoint
3479          { - 0.5 \arrayrulewidth - \arraycolsep }
3480          \c_zero_dim
3481      }
3482      { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3483  }
3484  \str_if_empty:NF \l_@@_name_str
3485  {
3486      \pgfnodealias
3487      { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3488      { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489  }
3490  \endpgfpicture

3491  \bool_if:NT \g_@@_last_col_found_bool
3492  {
3493      \hbox_overlap_right:n
3494      {
3495          \skip_horizontal:N \g_@@_width_last_col_dim
3496          \skip_horizontal:N \col@sep
3497          \bool_if:NT \l_@@_code_before_bool

```

```

3498         {
3499             \pgfsys@markposition
3500             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3501         }
3502         \pgfpicture
3503         \pgfrememberpicturepositiononpagetrue
3504         \pgfcoordinate
3505         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3506         \pgfpointorigin
3507         \str_if_empty:NF \l_@@_name_str
3508         {
3509             \pgfnodealias
3510             {
3511                 \l_@@_name_str - col
3512                 - \int_eval:n { \g_@@_col_total_int + 1 }
3513             }
3514             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3515         }
3516         \endpgfpicture
3517     }
3518 }
3519 \cr
3520 }

```

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

```

3521 \tl_const:Nn \c_@@_preamble_first_col_tl
3522 {
3523     >
3524     {

```

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).

```

3525         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3526         \bool_gset_true:N \g_@@_after_col_zero_bool
3527         \@@_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.

```

3528         \hbox_set:Nw \l_@@_cell_box
3529         \@@_math_toggle:
3530         \@@_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”.

```

3531         \int_compare:nNnT \c@iRow > \c_zero_int
3532         {
3533             \bool_lazy_or:nnT
3534             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3535             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3536             {
3537                 \l_@@_code_for_first_col_tl
3538                 \xglobal \colorlet { nicematrix-first-col } { . }
3539             }
3540         }
3541     }

```

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`.

```

3542     l
3543     <
3544     {
3545         \@@_math_toggle:
3546         \hbox_set_end:

```



```

3547 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3548 \@@_adjust_size_box:
3549 \@@_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.

```

3550 \dim_gset:Nn \g_@@_width_first_col_dim
3551 { \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.

```

3552 \hbox_overlap_left:n
3553 {
3554   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3555     \@@_node_for_cell:
3556     { \box_use_drop:N \l_@@_cell_box }
3557     \skip_horizontal:N \l_@@_left_delim_dim
3558     \skip_horizontal:N \l_@@_left_margin_dim
3559     \skip_horizontal:N \l_@@_extra_left_margin_dim
3560   }
3561   \bool_gset_false:N \g_@@_empty_cell_bool
3562   \skip_horizontal:N -2\col@sep
3563 }
3564 }

```

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

```

3565 \tl_const:Nn \c_@@_preamble_last_col_tl
3566 {
3567   >
3568   {
3569     \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).

```

3570 \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.

```

3571 \bool_gset_true:N \g_@@_last_col_found_bool
3572 \int_gincr:N \c@jCol
3573 \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.

```

3574 \hbox_set:Nw \l_@@_cell_box
3575 \@@_math_toggle:
3576 \@@_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”.

```

3577 \int_compare:nNnT \c@iRow > \c_zero_int
3578 {
3579   \bool_lazy_or:nnT
3580   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3581   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3582   {
3583     \l_@@_code_for_last_col_tl
3584     \xglobal \colorlet { nicematrix-last-col } { . }
3585   }
3586 }
3587 }
3588 1
3589 <
3590 {
3591   \@@_math_toggle:
3592   \hbox_set_end:
3593   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3594   \@@_adjust_size_box:
3595   \@@_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.

```

3596 \dim_gset:Nn \g_@@_width_last_col_dim
3597 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3598 \skip_horizontal:N -2\col@sep

```

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

```

3599 \hbox_overlap_right:n
3600 {
3601   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3602   {
3603     \skip_horizontal:N \l_@@_right_delim_dim
3604     \skip_horizontal:N \l_@@_right_margin_dim
3605     \skip_horizontal:N \l_@@_extra_right_margin_dim
3606     \@@_node_for_cell:
3607   }
3608 }
3609 \bool_gset_false:N \g_@@_empty_cell_bool
3610 }
3611 }

```

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

```

3612 \NewDocumentEnvironment { NiceArray } { }
3613 {
3614   \bool_gset_false:N \g_@@_delims_bool
3615   \str_if_empty:NT \g_@@_name_env_str
3616   { \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).

```

3617   \NiceArrayWithDelims . .
3618 }
3619 { \endNiceArrayWithDelims }

```

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

```

3620 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3621 {
3622   \NewDocumentEnvironment { #1 NiceArray } { }
3623   {
3624     \bool_gset_true:N \g_@@_delims_bool
3625     \str_if_empty:NT \g_@@_name_env_str
3626     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3627     \@@_test_if_math_mode:
3628     \NiceArrayWithDelims #2 #3
3629   }
3630   { \endNiceArrayWithDelims }
3631 }
3632 \@@_def_env:nnn p ( )
3633 \@@_def_env:nnn b [ ]
3634 \@@_def_env:nnn B \{ \}
3635 \@@_def_env:nnn v | |
3636 \@@_def_env:nnn V \| \|

```

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

```

3637 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3638 {
3639   \bool_set_false:N \l_@@_preamble_bool
3640   \tl_clear:N \l_tmpa_tl
3641   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3642     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3643   \tl_put_right:Nn \l_tmpa_tl
3644     {
3645       *
3646       {
3647         \int_case:nnF \l_@@_last_col_int
3648           {
3649             { -2 } { \c@MaxMatrixCols }
3650             { -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).

```

3651       }
3652       { \int_eval:n { \l_@@_last_col_int - 1 } }
3653     }
3654     { #2 }
3655   }
3656   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3657   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3658 }
3659 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3660 \clist_map_inline:nn { p , b , B , v , V }
3661 {
3662   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3663   {
3664     \bool_gset_true:N \g_@@_delims_bool
3665     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3666     \int_if_zero:nT \l_@@_last_col_int
3667       {
3668         \bool_set_true:N \l_@@_last_col_without_value_bool
3669         \int_set:Nn \l_@@_last_col_int { -1 }
3670       }
3671     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3672     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3673   }
3674   { \use:c { end #1 NiceArray } }
3675 }

```

We define also an environment `{NiceMatrix}`

```

3676 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3677 {
3678   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3679   \int_if_zero:nT \l_@@_last_col_int
3680     {
3681       \bool_set_true:N \l_@@_last_col_without_value_bool
3682       \int_set:Nn \l_@@_last_col_int { -1 }
3683     }
3684   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3685   \bool_lazy_or:nnT
3686     { \clist_if_empty_p:N \l_@@_vlines_clist }
3687     { \l_@@_except_borders_bool }
3688     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3689   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3690 }
3691 { \endNiceArray }

```

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

```
3692 \cs_new_protected:Npn \@@_NotEmpty:
3693 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3694 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3695 {
```

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`.

```
3696   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3697     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3698   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3699   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3700   \tl_if_empty:NF \l_@@_short_caption_tl
3701   {
3702     \tl_if_empty:NT \l_@@_caption_tl
3703     {
3704       \@@_error_or_warning:n { short-caption-without-caption }
3705       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3706     }
3707   }
3708   \tl_if_empty:NF \l_@@_label_tl
3709   {
3710     \tl_if_empty:NT \l_@@_caption_tl
3711     { \@@_error_or_warning:n { label-without-caption } }
3712   }
3713   \NewDocumentEnvironment { TabularNote } { b }
3714   {
3715     \bool_if:NTF \l_@@_in_code_after_bool
3716       { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3717     {
3718       \tl_if_empty:NF \g_@@_tabularnote_tl
3719       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3720       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3721     }
3722   }
3723   { }
3724   \@@_settings_for_tabular:
3725   \NiceArray { #2 }
3726 }
3727 {
3728   \endNiceArray
3729   \bool_if:NT \c_@@_tagging_array_bool
3730     { \UseTaggingSocket { tbl / hmode / end } }
3731 }
3732 \cs_new_protected:Npn \@@_settings_for_tabular:
3733 {
3734   \bool_set_true:N \l_@@_tabular_bool
3735   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3736   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3737   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3738 }
```

```
3739 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3740 {
3741   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3742   \dim_zero_new:N \l_@@_width_dim
3743   \dim_set:Nn \l_@@_width_dim { #1 }
3744   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3745   \@@_settings_for_tabular:
```

```

3746 \NiceArray { #3 }
3747 }
3748 {
3749 \endNiceArray
3750 \int_if_zero:nT \g_@@_total_X_weight_int
3751 { \@@_error:n { NiceTabularX~without~X } }
3752 }

3753 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3754 {
3755 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3756 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3757 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3758 \@@_settings_for_tabular:
3759 \NiceArray { #3 }
3760 }
3761 { \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`).

```

3762 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3763 {
3764 \bool_lazy_all:nT
3765 {
3766 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3767 \l_@@_hvlines_bool
3768 { ! \g_@@_delims_bool }
3769 { ! \l_@@_except_borders_bool }
3770 }
3771 {
3772 \bool_set_true:N \l_@@_except_borders_bool
3773 \clist_if_empty:NF \l_@@_corners_clist
3774 { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3775 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3776 {
3777 \@@_stroke_block:nnn
3778 {
3779 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3780 draw = \l_@@_rules_color_tl
3781 }
3782 { 1-1 }
3783 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3784 }
3785 }
3786 }

3787 \cs_new_protected:Npn \@@_after_array:
3788 {

```

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

```

3789 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3790 \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.

```

3791     \bool_if:NT \g_@@_last_col_found_bool
3792     { \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`.

```

3793     \bool_if:NT \l_@@_last_col_without_value_bool
3794     { \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.

```

3795     \bool_if:NT \l_@@_last_row_without_value_bool
3796     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3797     \tl_gput_right:Nx \g_@@_aux_tl
3798     {
3799       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3800       {
3801         \int_use:N \l_@@_first_row_int ,
3802         \int_use:N \c@iRow ,
3803         \int_use:N \g_@@_row_total_int ,
3804         \int_use:N \l_@@_first_col_int ,
3805         \int_use:N \c@jCol ,
3806         \int_use:N \g_@@_col_total_int
3807       }
3808     }

```

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`).

```

3809     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3810     {
3811       \tl_gput_right:Nx \g_@@_aux_tl
3812       {
3813         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3814         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3815       }
3816     }
3817     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3818     {
3819       \tl_gput_right:Nx \g_@@_aux_tl
3820       {
3821         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3822         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3823         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3824         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3825       }
3826     }

```

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

```

3827     \@@_create_diag_nodes:

```

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

```

3828     \pgfpicture
3829     \int_step_inline:nn \c@iRow
3830     {
3831       \pgfnodealias
3832       { \@@_env: - ##1 - last }
3833       { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3834     }
3835     \int_step_inline:nn \c@jCol
3836     {
3837         \pgfnodealias
3838         { \@@_env: - last - ##1 }
3839         { \@@_env: - \int_use:N \c@iRow - ##1 }
3840     }
3841     \str_if_empty:NF \l_@@_name_str
3842     {
3843         \int_step_inline:nn \c@iRow
3844         {
3845             \pgfnodealias
3846             { \l_@@_name_str - ##1 - last }
3847             { \@@_env: - ##1 - \int_use:N \c@jCol }
3848         }
3849         \int_step_inline:nn \c@jCol
3850         {
3851             \pgfnodealias
3852             { \l_@@_name_str - last - ##1 }
3853             { \@@_env: - \int_use:N \c@iRow - ##1 }
3854         }
3855     }
3856     \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.

```

3857     \bool_if:NT \l_@@_parallelize_diags_bool
3858     {
3859         \int_gzero_new:N \g_@@_ddots_int
3860         \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.

```

3861         \dim_gzero_new:N \g_@@_delta_x_one_dim
3862         \dim_gzero_new:N \g_@@_delta_y_one_dim
3863         \dim_gzero_new:N \g_@@_delta_x_two_dim
3864         \dim_gzero_new:N \g_@@_delta_y_two_dim
3865     }

3866     \int_zero_new:N \l_@@_initial_i_int
3867     \int_zero_new:N \l_@@_initial_j_int
3868     \int_zero_new:N \l_@@_final_i_int
3869     \int_zero_new:N \l_@@_final_j_int
3870     \bool_set_false:N \l_@@_initial_open_bool
3871     \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.

```

3872     \bool_if:NT \l_@@_small_bool
3873     {
3874         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3875         \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.

```

3876         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3877         { 0.6 \l_@@_xdots_shorten_start_dim }
3878         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

---

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

```

3879         { 0.6 \l_@@_xdots_shorten_end_dim }
3880     }

```

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

```

3881 \@@_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.

```

3882 \@@_compute_corners:

```

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

```

3883 \@@_adjust_pos_of_blocks_seq:
3884 \@@_deal_with_rounded_corners:
3885 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3886 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3887 \IfPackageLoadedTF { tikz }
3888 {
3889     \tikzset
3890     {
3891         every-picture / .style =
3892         {
3893             overlay ,
3894             remember~picture ,
3895             name~prefix = \@@_env: -
3896         }
3897     }
3898 }
3899 { }
3900 \bool_if:NT \c_@@_tagging_array_bool
3901 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3902 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3903 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3904 \cs_set_eq:NN \OverBrace \@@_OverBrace
3905 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3906 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3907 \cs_set_eq:NN \line \@@_line
3908 \g_@@_pre_code_after_tl
3909 \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.

```

3910 \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).

```

3911 \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`).

```

3912 % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3913 % { \@@_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:`.

```

3914 \bool_set_true:N \l_@@_in_code_after_bool
3915 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3916 \scan_stop:
3917 \tl_gclear:N \g_nicematrix_code_after_tl
3918 \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.

```

3919 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3920 \tl_if_empty:NF \g_@@_pre_code_before_tl
3921 {
3922   \tl_gput_right:Nx \g_@@_aux_tl
3923   {
3924     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3925     { \exp_not:o \g_@@_pre_code_before_tl }
3926   }
3927   \tl_gclear:N \g_@@_pre_code_before_tl
3928 }
3929 \tl_if_empty:NF \g_nicematrix_code_before_tl
3930 {
3931   \tl_gput_right:Nx \g_@@_aux_tl
3932   {
3933     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3934     { \exp_not:o \g_nicematrix_code_before_tl }
3935   }
3936   \tl_gclear:N \g_nicematrix_code_before_tl
3937 }

3938 \str_gclear:N \g_@@_name_env_str
3939 \@@_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`.

```

3940 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3941 }

```

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`.

```

3942 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3943 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format  $i$ - $j$ . However, the user is allowed to omit  $i$  or  $j$  (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

3944 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3945 {

```

---

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

```

3946 \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3947 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3948 }

```

The following command must *not* be protected.

```

3949 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3950 {
3951   { #1 }
3952   { #2 }
3953   {
3954     \int_compare:nNnTF { #3 } > { 99 }
3955     { \int_use:N \c@iRow }
3956     { #3 }
3957   }
3958   {
3959     \int_compare:nNnTF { #4 } > { 99 }
3960     { \int_use:N \c@jCol }
3961     { #4 }
3962   }
3963   { #5 }
3964 }

```

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).

```

3965 \hook_gput_code:nnn { begindocument } { . }
3966 {
3967   \cs_new_protected:Npx \@@_draw_dotted_lines:
3968   {
3969     \c_@@_pgfortikzpicture_tl
3970     \@@_draw_dotted_lines_i:
3971     \c_@@_endpgfortikzpicture_tl
3972   }
3973 }

```

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

```

3974 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3975 {
3976   \pgfrememberpicturerepositiononpagetrue
3977   \pgfrelevantforpicturesizefalse
3978   \g_@@_HVdotsfor_lines_tl
3979   \g_@@_Vdots_lines_tl
3980   \g_@@_Ddots_lines_tl
3981   \g_@@_Iddots_lines_tl
3982   \g_@@_Cdots_lines_tl
3983   \g_@@_Ldots_lines_tl
3984 }

3985 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3986 {
3987   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3988   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3989 }

```

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

```

3990 \pgfdeclareshape { @@_diag_node }
3991 {
3992   \savedanchor { \five }
3993   {
3994     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

3995         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3996     }
3997     \anchor { 5 } { \five }
3998     \anchor { center } { \pgfpointorigin }
3999 }

```

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).

```

4000 \cs_new_protected:Npn \@@_create_diag_nodes:
4001 {
4002     \pgfpicture
4003     \pgfrememberpicturepositiononpagetrue
4004     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4005     {
4006         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4007         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4008         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4009         \dim_set_eq:NN \l_tmpb_dim \pgf@y
4010         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4011         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4012         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4013         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4014         \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.

```

4015         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4016         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4017         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4018         \str_if_empty:NF \l_@@_name_str
4019         { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4020     }

```

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

```

4021     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4022     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4023     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4024     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4025     \pgfcoordinate
4026     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4027     \pgfnodealias
4028     { \@@_env: - last }
4029     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4030     \str_if_empty:NF \l_@@_name_str
4031     {
4032         \pgfnodealias
4033         { \l_@@_name_str - \int_use:N \l_tmpa_int }
4034         { \@@_env: - \int_use:N \l_tmpa_int }
4035         \pgfnodealias
4036         { \l_@@_name_str - last }
4037         { \@@_env: - last }
4038     }
4039     \endpgfpicture
4040 }

```

## 17 We draw the dotted lines

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

its right.

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

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

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

This command computes:

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

```
4041 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4042 {
```

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

```
4043 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4044 \int_set:Nn \l_@@_initial_i_int { #1 }
4045 \int_set:Nn \l_@@_initial_j_int { #2 }
4046 \int_set:Nn \l_@@_final_i_int { #1 }
4047 \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.

```
4048 \bool_set_false:N \l_@@_stop_loop_bool
4049 \bool_do_until:Nn \l_@@_stop_loop_bool
4050 {
4051   \int_add:Nn \l_@@_final_i_int { #3 }
4052   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4053   \bool_set_false:N \l_@@_final_open_bool
4054   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4055   {
4056     \int_compare:nNnTF { #3 } = \c_one_int
4057     { \bool_set_true:N \l_@@_final_open_bool }
4058     {
4059       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4060       { \bool_set_true:N \l_@@_final_open_bool }
4061     }
4062   }
4063   {
4064     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4065     {
4066       \int_compare:nNnT { #4 } = { -1 }
4067       { \bool_set_true:N \l_@@_final_open_bool }
4068     }
4069     {
```

```

4070         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4071         {
4072             \int_compare:nNnT { #4 } = \c_one_int
4073             { \bool_set_true:N \l_@@_final_open_bool }
4074         }
4075     }
4076 }
4077 \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.

```

4078 {

```

We do a step backwards.

```

4079     \int_sub:Nn \l_@@_final_i_int { #3 }
4080     \int_sub:Nn \l_@@_final_j_int { #4 }
4081     \bool_set_true:N \l_@@_stop_loop_bool
4082 }

```

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`.

```

4083 {
4084     \cs_if_exist:cTF
4085     {
4086         @@ _ dotted _
4087         \int_use:N \l_@@_final_i_int -
4088         \int_use:N \l_@@_final_j_int
4089     }
4090     {
4091         \int_sub:Nn \l_@@_final_i_int { #3 }
4092         \int_sub:Nn \l_@@_final_j_int { #4 }
4093         \bool_set_true:N \l_@@_final_open_bool
4094         \bool_set_true:N \l_@@_stop_loop_bool
4095     }
4096     {
4097         \cs_if_exist:cTF
4098         {
4099             pgf @ sh @ ns @ \@@_env:
4100             - \int_use:N \l_@@_final_i_int
4101             - \int_use:N \l_@@_final_j_int
4102         }
4103         { \bool_set_true:N \l_@@_stop_loop_bool }
4104     }
4105 }

```

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.

```

4104 {
4105     \cs_set:cpn
4106     {
4107         @@ _ dotted _
4108         \int_use:N \l_@@_final_i_int -
4109         \int_use:N \l_@@_final_j_int
4110     }
4111     { }
4112 }
4113 }
4114 }
4115 }

```

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

```

4116 \bool_set_false:N \l_@@_stop_loop_bool

```

```

4117 \bool_do_until:Nn \l_@@_stop_loop_bool
4118 {
4119   \int_sub:Nn \l_@@_initial_i_int { #3 }
4120   \int_sub:Nn \l_@@_initial_j_int { #4 }
4121   \bool_set_false:N \l_@@_initial_open_bool
4122   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4123   {
4124     \int_compare:nNnTF { #3 } = \c_one_int
4125     { \bool_set_true:N \l_@@_initial_open_bool }
4126     {
4127       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4128       { \bool_set_true:N \l_@@_initial_open_bool }
4129     }
4130   }
4131   {
4132     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4133     {
4134       \int_compare:nNnT { #4 } = \c_one_int
4135       { \bool_set_true:N \l_@@_initial_open_bool }
4136     }
4137     {
4138       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4139       {
4140         \int_compare:nNnT { #4 } = { -1 }
4141         { \bool_set_true:N \l_@@_initial_open_bool }
4142       }
4143     }
4144   }
4145   \bool_if:NNTF \l_@@_initial_open_bool
4146   {
4147     \int_add:Nn \l_@@_initial_i_int { #3 }
4148     \int_add:Nn \l_@@_initial_j_int { #4 }
4149     \bool_set_true:N \l_@@_stop_loop_bool
4150   }
4151   {
4152     \cs_if_exist:cTF
4153     {
4154       @@ _ dotted _
4155       \int_use:N \l_@@_initial_i_int -
4156       \int_use:N \l_@@_initial_j_int
4157     }
4158     {
4159       \int_add:Nn \l_@@_initial_i_int { #3 }
4160       \int_add:Nn \l_@@_initial_j_int { #4 }
4161       \bool_set_true:N \l_@@_initial_open_bool
4162       \bool_set_true:N \l_@@_stop_loop_bool
4163     }
4164     {
4165       \cs_if_exist:cTF
4166       {
4167         pgf @ sh @ ns @ \@@_env:
4168         - \int_use:N \l_@@_initial_i_int
4169         - \int_use:N \l_@@_initial_j_int
4170       }
4171       { \bool_set_true:N \l_@@_stop_loop_bool }
4172       {
4173         \cs_set:cpn
4174         {
4175           @@ _ dotted _
4176           \int_use:N \l_@@_initial_i_int -
4177           \int_use:N \l_@@_initial_j_int
4178         }
4179         { }

```

```

4180     }
4181   }
4182 }
4183 }

```

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.

```

4184   \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4185   {
4186     { \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`.

```

4187     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4188     { \int_use:N \l_@@_final_i_int }
4189     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4190     { } % for the name of the block
4191   }
4192 }

```

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.

```

4193 \cs_new_protected:Npn \@@_open_shorten:
4194 {
4195   \bool_if:NT \l_@@_initial_open_bool
4196   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4197   \bool_if:NT \l_@@_final_open_bool
4198   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4199 }

```

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).

```

4200 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4201 {
4202   \int_set:Nn \l_@@_row_min_int 1
4203   \int_set:Nn \l_@@_col_min_int 1
4204   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4205   \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`.

```

4206   \seq_map_inline:Nn \g_@@_submatrix_seq
4207   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4208 }

```

#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.

```

4209 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4210 {
4211   \int_compare:nNnF { #3 } > { #1 }
4212   {
4213     \int_compare:nNnF { #1 } > { #5 }
4214     {
4215       \int_compare:nNnF { #4 } > { #2 }
4216       {
4217         \int_compare:nNnF { #2 } > { #6 }

```

```

4218         {
4219             \int_set:Nn \l_@@_row_min_int
4220             { \int_max:nn \l_@@_row_min_int { #3 } }
4221             \int_set:Nn \l_@@_col_min_int
4222             { \int_max:nn \l_@@_col_min_int { #4 } }
4223             \int_set:Nn \l_@@_row_max_int
4224             { \int_min:nn \l_@@_row_max_int { #5 } }
4225             \int_set:Nn \l_@@_col_max_int
4226             { \int_min:nn \l_@@_col_max_int { #6 } }
4227         }
4228     }
4229 }
4230 }
4231 }

4232 \cs_new_protected:Npn \@@_set_initial_coords:
4233 {
4234     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4235     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4236 }
4237 \cs_new_protected:Npn \@@_set_final_coords:
4238 {
4239     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4240     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4241 }
4242 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4243 {
4244     \pgfpointanchor
4245     {
4246         \@@_env:
4247         - \int_use:N \l_@@_initial_i_int
4248         - \int_use:N \l_@@_initial_j_int
4249     }
4250     { #1 }
4251     \@@_set_initial_coords:
4252 }
4253 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4254 {
4255     \pgfpointanchor
4256     {
4257         \@@_env:
4258         - \int_use:N \l_@@_final_i_int
4259         - \int_use:N \l_@@_final_j_int
4260     }
4261     { #1 }
4262     \@@_set_final_coords:
4263 }
4264 \cs_new_protected:Npn \@@_open_x_initial_dim:
4265 {
4266     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4267     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4268     {
4269         \cs_if_exist:cT
4270         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4271         {
4272             \pgfpointanchor
4273             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4274             { west }
4275             \dim_set:Nn \l_@@_x_initial_dim
4276             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4277         }
4278     }

```

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



```

4279 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4280 {
4281   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4282   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4283   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4284 }
4285 }
4286 \cs_new_protected:Npn \@@_open_x_final_dim:
4287 {
4288   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4289   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4290   {
4291     \cs_if_exist:cT
4292     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4293     {
4294       \pgfpointanchor
4295       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4296       { east }
4297       \dim_set:Nn \l_@@_x_final_dim
4298       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4299     }
4300   }

```

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

```

4301 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4302 {
4303   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4304   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4305   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4306 }
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_Ldots: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_Ldots:
4325 \group_end:
4326 }
4327 }

```

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.

```

4328 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4329 {
4330   \bool_if:NTF \l_@@_initial_open_bool
4331   {
4332     \@@_open_x_initial_dim:
4333     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4334     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4335   }
4336   { \@@_set_initial_coords_from_anchor:n { base-east } }
4337   \bool_if:NTF \l_@@_final_open_bool
4338   {
4339     \@@_open_x_final_dim:
4340     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4341     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4342   }
4343   { \@@_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.

```

4344   \bool_lazy_all:nTF
4345   {
4346     \l_@@_initial_open_bool
4347     \l_@@_final_open_bool
4348     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4349   }
4350   {
4351     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4352     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4353   }

```

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 (?).

```

4354   {
4355     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4356     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4357   }
4358   \@@_draw_line:
4359 }

```

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

```

4360 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4361 {
4362   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4363   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4364   {
4365     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 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.

```

4366     \group_begin:
4367     \@@_open_shorten:
4368     \int_if_zero:nTF { #1 }
4369     { \color { nicematrix-first-row } }
4370     {

```

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.

```

4371         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4372         { \color { nicematrix-last-row } }
4373     }
4374     \keys_set:nn { NiceMatrix / xdots } { #3 }
4375     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4376     \@@_actually_draw_Cdots:
4377 \group_end:
4378 }
4379 }

```

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`.

```

4380 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4381 {
4382     \bool_if:NTF \l_@@_initial_open_bool
4383     { \@@_open_x_initial_dim: }
4384     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4385     \bool_if:NTF \l_@@_final_open_bool
4386     { \@@_open_x_final_dim: }
4387     { \@@_set_final_coords_from_anchor:n { mid-west } }
4388     \bool_lazy_and:nnTF
4389     \l_@@_initial_open_bool
4390     \l_@@_final_open_bool
4391     {
4392         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4393         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4394         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4395         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4396         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4397     }
4398     {
4399         \bool_if:NT \l_@@_initial_open_bool
4400         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4401         \bool_if:NT \l_@@_final_open_bool
4402         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4403     }
4404     \@@_draw_line:
4405 }
4406 \cs_new_protected:Npn \@@_open_y_initial_dim:
4407 {
4408     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4409     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4410     {

```

```

4411 \cs_if_exist:cT
4412 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4413 {
4414     \pgfpointanchor
4415     { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4416     { north }
4417     \dim_set:Nn \l_@@_y_initial_dim
4418     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4419 }
4420 }
4421 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4422 {
4423     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4424     \dim_set:Nn \l_@@_y_initial_dim
4425     {
4426         \fp_to_dim:n
4427         {
4428             \pgf@y
4429             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4430         }
4431     }
4432 }
4433 }
4434 \cs_new_protected:Npn \@@_open_y_final_dim:
4435 {
4436     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4437     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4438     {
4439         \cs_if_exist:cT
4440         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4441         {
4442             \pgfpointanchor
4443             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4444             { south }
4445             \dim_set:Nn \l_@@_y_final_dim
4446             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4447         }
4448     }
4449     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4450     {
4451         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4452         \dim_set:Nn \l_@@_y_final_dim
4453         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4454     }
4455 }

```

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.

```

4456 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4457 {
4458     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4459     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4460     {
4461         \@@_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.

```

4462 \group_begin:
4463     \@@_open_shorten:
4464     \int_if_zero:nTF { #2 }
4465     { \color { nicematrix-first-col } }
4466     {
4467         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4468         { \color { nicematrix-last-col } }

```

```

4469     }
4470     \keys_set:nn { NiceMatrix / xdots } { #3 }
4471     \tl_if_empty:oF \l_@@_xdots_color_tl
4472     { \color { \l_@@_xdots_color_tl } }
4473     \@@_actually_draw_Vdots:
4474     \group_end:
4475   }
4476 }

```

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`.

```

4477 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4478 {

```

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

```

4479     \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.

```

4480     {
4481       \@@_open_y_initial_dim:
4482       \@@_open_y_final_dim:
4483       \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4484     {
4485       \@@_qpoint:n { col - 1 }
4486       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4487       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4488       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4489       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4490     }
4491     {
4492       \bool_lazy_and:nnTF
4493       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4494       { \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”.

```

4495     {
4496       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4497       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4498       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4499       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4500       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4501     }

```

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

```

4502     {
4503       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4504       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4505       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4506       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4507     }
4508   }
4509 }

```

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.

```

4510     {
4511         \bool_set_false:N \l_tmpa_bool
4512         \bool_if:NF \l_@@_initial_open_bool
4513         {
4514             \bool_if:NF \l_@@_final_open_bool
4515             {
4516                 \@@_set_initial_coords_from_anchor:n { south-west }
4517                 \@@_set_final_coords_from_anchor:n { north-west }
4518                 \bool_set:Nn \l_tmpa_bool
4519                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4520             }
4521         }

```

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

```

4522     \bool_if:NTF \l_@@_initial_open_bool
4523     {
4524         \@@_open_y_initial_dim:
4525         \@@_set_final_coords_from_anchor:n { north }
4526         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4527     }
4528     {
4529         \@@_set_initial_coords_from_anchor:n { south }
4530         \bool_if:NTF \l_@@_final_open_bool
4531         \@@_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.

```

4532     {
4533         \@@_set_final_coords_from_anchor:n { north }
4534         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4535         {
4536             \dim_set:Nn \l_@@_x_initial_dim
4537             {
4538                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4539                 \l_@@_x_initial_dim \l_@@_x_final_dim
4540             }
4541         }
4542     }
4543 }
4544 }
4545 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4546 \@@_draw_line:
4547 }

```

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.

```

4548 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4549 {
4550     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4551     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4552     {
4553         \@@_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.

```

4554     \group_begin:
4555     \@@_open_shorten:

```

```

4556         \keys_set:nn { NiceMatrix / xdots } { #3 }
4557         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4558         \@@_actually_draw_Ddots:
4559     \group_end:
4560 }
4561 }

```

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.`

```

4562 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4563 {
4564     \bool_if:NTF \l_@@_initial_open_bool
4565     {
4566         \@@_open_y_initial_dim:
4567         \@@_open_x_initial_dim:
4568     }
4569     { \@@_set_initial_coords_from_anchor:n { south~east } }
4570 \bool_if:NTF \l_@@_final_open_bool
4571 {
4572     \@@_open_x_final_dim:
4573     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4574 }
4575 { \@@_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.

```

4576     \bool_if:NT \l_@@_parallelize_diags_bool
4577     {
4578         \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).

```

4579         \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.

```

4580     {
4581         \dim_gset:Nn \g_@@_delta_x_one_dim
4582         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4583         \dim_gset:Nn \g_@@_delta_y_one_dim
4584         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4585     }

```

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`.

```

4586     {
4587         \dim_set:Nn \l_@@_y_final_dim
4588         {
4589             \l_@@_y_initial_dim +
4590             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4591             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4592         }
4593     }
4594 }
4595 \@@_draw_line:
4596 }

```

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.

```

4597 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4598 {
4599   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4600   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4601   {
4602     \@@_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.

```

4603   \group_begin:
4604   \@@_open_shorten:
4605   \keys_set:nn { NiceMatrix / xdots } { #3 }
4606   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4607   \@@_actually_draw_Iddots:
4608   \group_end:
4609 }
4610 }

```

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`.

```

4611 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4612 {
4613   \bool_if:NTF \l_@@_initial_open_bool
4614   {
4615     \@@_open_y_initial_dim:
4616     \@@_open_x_initial_dim:
4617   }
4618   { \@@_set_initial_coords_from_anchor:n { south~west } }
4619   \bool_if:NTF \l_@@_final_open_bool
4620   {
4621     \@@_open_y_final_dim:
4622     \@@_open_x_final_dim:
4623   }
4624   { \@@_set_final_coords_from_anchor:n { north~east } }
4625   \bool_if:NT \l_@@_parallelize_diags_bool
4626   {
4627     \int_gincr:N \g_@@_iddots_int
4628     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4629     {
4630       \dim_gset:Nn \g_@@_delta_x_two_dim
4631       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4632       \dim_gset:Nn \g_@@_delta_y_two_dim
4633       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4634     }
4635     {
4636       \dim_set:Nn \l_@@_y_final_dim
4637       {
4638         \l_@@_y_initial_dim +
4639         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4640         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim

```



```

4641         }
4642     }
4643 }
4644 \@@_draw_line:
4645 }

```

## 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`

```

4646 \cs_new_protected:Npn \@@_draw_line:
4647 {
4648     \pgfrememberpicturepositiononpagetrue
4649     \pgf@relevantforpicturesizefalse
4650     \bool_lazy_or:nnTF
4651     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4652     \l_@@_dotted_bool
4653     \@@_draw_standard_dotted_line:
4654     \@@_draw_unstandard_dotted_line:
4655 }

```

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.

```

4656 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4657 {
4658     \begin { scope }
4659     \@@_draw_unstandard_dotted_line:o
4660     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4661 }

```

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.

```

4662 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4663 {
4664     \@@_draw_unstandard_dotted_line:nooo
4665     { #1 }
4666     \l_@@_xdots_up_tl
4667     \l_@@_xdots_down_tl
4668     \l_@@_xdots_middle_tl
4669 }
4670 \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.

```

4671 \hook_gput_code:nnn { begindocument } { . }
4672 {
4673   \IfPackageLoadedTF { tikz }
4674   {
4675     \tikzset
4676     {
4677       @@_node_above / .style = { sloped , above } ,
4678       @@_node_below / .style = { sloped , below } ,
4679       @@_node_middle / .style =
4680       {
4681         sloped ,
4682         inner-sep = \c_@@_innersep_middle_dim
4683       }
4684     }
4685   }
4686 }
4687 }

4688 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4689 {

```

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.

```

4690 \dim_zero_new:N \l_@@_l_dim
4691 \dim_set:Nn \l_@@_l_dim
4692 {
4693   \fp_to_dim:n
4694   {
4695     sqrt
4696     (
4697       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4698       +
4699       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4700     )
4701   }
4702 }

```

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.

```

4703 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4704 {
4705   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4706   \@@_draw_unstandard_dotted_line_i:
4707 }

```

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

```

4708 \bool_if:NT \l_@@_xdots_h_labels_bool
4709 {
4710   \tikzset
4711   {
4712     @@_node_above / .style = { auto = left } ,
4713     @@_node_below / .style = { auto = right } ,
4714     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4715   }
4716 }

```

```

4717 \tl_if_empty:nF { #4 }
4718 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4719 \draw
4720 [ #1 ]
4721 ( \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).

```

4722 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4723 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4724 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4725 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4726 \end { scope }
4727 }

4728 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4729 {
4730 \dim_set:Nn \l_tmpa_dim
4731 {
4732 \l_@@_x_initial_dim
4733 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4734 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4735 }
4736 \dim_set:Nn \l_tmpb_dim
4737 {
4738 \l_@@_y_initial_dim
4739 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4740 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4741 }
4742 \dim_set:Nn \l_@@_tmpc_dim
4743 {
4744 \l_@@_x_final_dim
4745 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4746 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4747 }
4748 \dim_set:Nn \l_@@_tmpd_dim
4749 {
4750 \l_@@_y_final_dim
4751 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4752 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4753 }
4754 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4755 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4756 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4757 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4758 }

4759 \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).

```

4760 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4761 {
4762 \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.

```

4763 \dim_zero_new:N \l_@@_l_dim
4764 \dim_set:Nn \l_@@_l_dim
4765 {
4766 \fp_to_dim:n
4767 {
4768 sqrt
4769 (

```

```

4770      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4771      +
4772      ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4773    )
4774  }
4775 }

```

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.

```

4776   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4777   {
4778     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4779     \@@_draw_standard_dotted_line_i:
4780   }
4781 \group_end:
4782 \bool_lazy_all:nF
4783 {
4784   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4785   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4786   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4787 }
4788 \l_@@_labels_standard_dotted_line:
4789 }
4790 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4791 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4792 {

```

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

```

4793   \int_set:Nn \l_tmpa_int
4794   {
4795     \dim_ratio:nn
4796     {
4797       \l_@@_l_dim
4798       - \l_@@_xdots_shorten_start_dim
4799       - \l_@@_xdots_shorten_end_dim
4800     }
4801     \l_@@_xdots_inter_dim
4802   }

```

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

```

4803   \dim_set:Nn \l_tmpa_dim
4804   {
4805     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4806     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4807   }
4808   \dim_set:Nn \l_tmpb_dim
4809   {
4810     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4811     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4812   }

```

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.

```

4813   \dim_gadd:Nn \l_@@_x_initial_dim
4814   {
4815     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4816     \dim_ratio:nn
4817     {
4818       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int

```

```

4819         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4820     }
4821     { 2 \l_@@_l_dim }
4822 }
4823 \dim_gadd:Nn \l_@@_y_initial_dim
4824 {
4825     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4826     \dim_ratio:nn
4827     {
4828         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4829         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4830     }
4831     { 2 \l_@@_l_dim }
4832 }
4833 \pgf@relevantforpicturesizefalse
4834 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4835 {
4836     \pgfpathcircle
4837     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4838     { \l_@@_xdots_radius_dim }
4839     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4840     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4841 }
4842 \pgfusepathqfill
4843 }

4844 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4845 {
4846     \pgfscope
4847     \pgftransformshift
4848     {
4849         \pgfpointlineattime { 0.5 }
4850         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4851         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4852     }
4853     \fp_set:Nn \l_tmpa_fp
4854     {
4855         atand
4856         (
4857             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4858             \l_@@_x_final_dim - \l_@@_x_initial_dim
4859         )
4860     }
4861     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4862     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4863     \tl_if_empty:NF \l_@@_xdots_middle_tl
4864     {
4865         \begin { pgfscope }
4866         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4867         \pgfnode
4868         { rectangle }
4869         { center }
4870         {
4871             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4872             {
4873                 \c_math_toggle_token
4874                 \scriptstyle \l_@@_xdots_middle_tl
4875                 \c_math_toggle_token
4876             }
4877         }
4878         { }
4879         {
4880             \pgfsetfillcolor { white }

```

```

4881         \pgfusepath { fill }
4882     }
4883     \end { pgfscope }
4884 }
4885 \tl_if_empty:NF \l_@@_xdots_up_tl
4886 {
4887     \pgfnode
4888     { rectangle }
4889     { south }
4890     {
4891         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4892         {
4893             \c_math_toggle_token
4894             \scriptstyle \l_@@_xdots_up_tl
4895             \c_math_toggle_token
4896         }
4897     }
4898     { }
4899     { \pgfusepath { } }
4900 }
4901 \tl_if_empty:NF \l_@@_xdots_down_tl
4902 {
4903     \pgfnode
4904     { rectangle }
4905     { north }
4906     {
4907         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4908         {
4909             \c_math_toggle_token
4910             \scriptstyle \l_@@_xdots_down_tl
4911             \c_math_toggle_token
4912         }
4913     }
4914     { }
4915     { \pgfusepath { } }
4916 }
4917 \endpgfscope
4918 }

```

## 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.

```

4919 \hook_gput_code:nnn { begindocument } { . }
4920 {
4921     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4922     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4923     \cs_new_protected:Npn \@@_Ldots
4924     { \@@_collect_options:n { \@@_Ldots_i } }
4925     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4926     {
4927         \int_if_zero:nTF \c@jCol

```

```

4928     { \@@_error:nn { in~first~col } \Ldots }
4929     {
4930       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4931       { \@@_error:nn { in~last~col } \Ldots }
4932       {
4933         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4934         { #1 , down = #2 , up = #3 , middle = #4 }
4935       }
4936     }
4937     \bool_if:NF \l_@@_nullify_dots_bool
4938     { \phantom { \ensuremath { \@@_old_ldots } } }
4939     \bool_gset_true:N \g_@@_empty_cell_bool
4940   }

4941 \cs_new_protected:Npn \@@_Cdots
4942 { \@@_collect_options:n { \@@_Cdots_i } }
4943 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4944 {
4945   \int_if_zero:nTF \c@jCol
4946   { \@@_error:nn { in~first~col } \Cdots }
4947   {
4948     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4949     { \@@_error:nn { in~last~col } \Cdots }
4950     {
4951       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4952       { #1 , down = #2 , up = #3 , middle = #4 }
4953     }
4954   }
4955   \bool_if:NF \l_@@_nullify_dots_bool
4956   { \phantom { \ensuremath { \@@_old_cdots } } }
4957   \bool_gset_true:N \g_@@_empty_cell_bool
4958 }

4959 \cs_new_protected:Npn \@@_Vdots
4960 { \@@_collect_options:n { \@@_Vdots_i } }
4961 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4962 {
4963   \int_if_zero:nTF \c@iRow
4964   { \@@_error:nn { in~first~row } \Vdots }
4965   {
4966     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4967     { \@@_error:nn { in~last~row } \Vdots }
4968     {
4969       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4970       { #1 , down = #2 , up = #3 , middle = #4 }
4971     }
4972   }
4973   \bool_if:NF \l_@@_nullify_dots_bool
4974   { \phantom { \ensuremath { \@@_old_vdots } } }
4975   \bool_gset_true:N \g_@@_empty_cell_bool
4976 }

4977 \cs_new_protected:Npn \@@_Ddots
4978 { \@@_collect_options:n { \@@_Ddots_i } }
4979 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4980 {
4981   \int_case:nnF \c@iRow
4982   {
4983     0 { \@@_error:nn { in~first~row } \Ddots }
4984     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4985   }

```

```

4986 {
4987   \int_case:nnF \c@jCol
4988   {
4989     0 { \@@_error:nn { in~first~col } \Ddots }
4990     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4991   }
4992   {
4993     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4994     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4995     { #1 , down = #2 , up = #3 , middle = #4 }
4996   }
4997 }
4998 }
4999 \bool_if:NF \l_@@_nullify_dots_bool
5000 { \phantom { \ensuremath { \@@_old_ddots } } }
5001 \bool_gset_true:N \g_@@_empty_cell_bool
5002 }

5003 \cs_new_protected:Npn \@@_Iddots
5004 { \@@_collect_options:n { \@@_Iddots_i } }
5005 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5006 {
5007   \int_case:nnF \c@iRow
5008   {
5009     0 { \@@_error:nn { in~first~row } \Iddots }
5010     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5011   }
5012   {
5013     \int_case:nnF \c@jCol
5014     {
5015       0 { \@@_error:nn { in~first~col } \Iddots }
5016       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5017     }
5018     {
5019       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5020       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5021       { #1 , down = #2 , up = #3 , middle = #4 }
5022     }
5023   }
5024   \bool_if:NF \l_@@_nullify_dots_bool
5025   { \phantom { \ensuremath { \@@_old_iddots } } }
5026   \bool_gset_true:N \g_@@_empty_cell_bool
5027 }
5028 }

```

End of the \AddToHook.

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

```

5029 \keys_define:nn { NiceMatrix / Ddots }
5030 {
5031   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5032   draw-first .default:n = true ,
5033   draw-first .value_forbidden:n = true
5034 }

```

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

```

5035 \cs_new_protected:Npn \@@_Hspace:
5036 {
5037   \bool_gset_true:N \g_@@_empty_cell_bool
5038   \hspace
5039 }

```



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`.

```
5040 \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`.

```
5041 \cs_new:Npn \@@_Hdotsfor:
5042 {
5043   \bool_lazy_and:nnTF
5044     { \int_if_zero_p:n \c@jCol }
5045     { \int_if_zero_p:n \l_@@_first_col_int }
5046     {
5047       \bool_if:NTF \g_@@_after_col_zero_bool
5048       {
5049         \multicolumn { 1 } { c } { }
5050         \@@_Hdotsfor_i
5051       }
5052       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5053     }
5054   {
5055     \multicolumn { 1 } { c } { }
5056     \@@_Hdotsfor_i
5057   }
5058 }
```

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:`).

```
5059 \hook_gput_code:nnn { begindocument } { . }
5060 {
5061   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5062   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5063 }
```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```
5063 \cs_new_protected:Npn \@@_Hdotsfor_i
5064 { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5065 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5066 {
5067   \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5068   {
5069     \@@_Hdotsfor:nnnn
5070     { \int_use:N \c@iRow }
5071     { \int_use:N \c@jCol }
5072     { #2 }
5073     {
5074       #1 , #3 ,
5075       down = \exp_not:n { #4 } ,
5076       up = \exp_not:n { #5 } ,
5077       middle = \exp_not:n { #6 }
5078     }
5079   }
5080   \prg_replicate:nn { #2 - 1 }
5081   {
5082     &
5083     \multicolumn { 1 } { c } { }
5084     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5085   }
5086 }
5087 }
```

```

5088 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5089 {
5090   \bool_set_false:N \l_@@_initial_open_bool
5091   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5092   \int_set:Nn \l_@@_initial_i_int { #1 }
5093   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5094   \int_compare:nNnTF { #2 } = \c_one_int
5095   {
5096     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5097     \bool_set_true:N \l_@@_initial_open_bool
5098   }
5099   {
5100     \cs_if_exist:cTF
5101     {
5102       pgf @ sh @ ns @ \@@_env:
5103       - \int_use:N \l_@@_initial_i_int
5104       - \int_eval:n { #2 - 1 }
5105     }
5106     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5107     {
5108       \int_set:Nn \l_@@_initial_j_int { #2 }
5109       \bool_set_true:N \l_@@_initial_open_bool
5110     }
5111   }
5112   \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5113   {
5114     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5115     \bool_set_true:N \l_@@_final_open_bool
5116   }
5117   {
5118     \cs_if_exist:cTF
5119     {
5120       pgf @ sh @ ns @ \@@_env:
5121       - \int_use:N \l_@@_final_i_int
5122       - \int_eval:n { #2 + #3 }
5123     }
5124     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5125     {
5126       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5127       \bool_set_true:N \l_@@_final_open_bool
5128     }
5129   }
5130   \group_begin:
5131   \@@_open_shorten:
5132   \int_if_zero:nTF { #1 }
5133   { \color { nicematrix-first-row } }
5134   {
5135     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5136     { \color { nicematrix-last-row } }
5137   }
5138
5139   \keys_set:nn { NiceMatrix / xdots } { #4 }
5140   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5141   \@@_actually_draw_Ldots:
5142   \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).

```

5143 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5144 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5145 }

5146 \hook_gput_code:nnn { begindocument } { . }
5147 {
5148   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5149   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5150   \cs_new_protected:Npn \@@_Vdotsfor:
5151     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5152   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5153     {
5154       \bool_gset_true:N \g_@@_empty_cell_bool
5155       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5156         {
5157           \@@_Vdotsfor:nnnn
5158           { \int_use:N \c@iRow }
5159           { \int_use:N \c@jCol }
5160           { #2 }
5161           {
5162             #1 , #3 ,
5163             down = \exp_not:n { #4 } ,
5164             up = \exp_not:n { #5 } ,
5165             middle = \exp_not:n { #6 }
5166           }
5167         }
5168     }
5169 }

```

```

5170 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5171 {
5172   \bool_set_false:N \l_@@_initial_open_bool
5173   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5174 \int_set:Nn \l_@@_initial_j_int { #2 }
5175 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5176 \int_compare:nNnTF { #1 } = \c_one_int
5177 {
5178   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5179   \bool_set_true:N \l_@@_initial_open_bool
5180 }
5181 {
5182   \cs_if_exist:cTF
5183     {
5184       pgf @ sh @ ns @ \@@_env:
5185       - \int_eval:n { #1 - 1 }
5186       - \int_use:N \l_@@_initial_j_int
5187     }
5188     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5189     {
5190       \int_set:Nn \l_@@_initial_i_int { #1 }
5191       \bool_set_true:N \l_@@_initial_open_bool
5192     }
5193 }
5194 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5195 {
5196   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5197   \bool_set_true:N \l_@@_final_open_bool
5198 }
5199 {

```

```

5200     \cs_if_exist:cTF
5201     {
5202         pgf @ sh @ ns @ \@@_env:
5203         - \int_eval:n { #1 + #3 }
5204         - \int_use:N \l_@@_final_j_int
5205     }
5206     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5207     {
5208         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5209         \bool_set_true:N \l_@@_final_open_bool
5210     }
5211 }
5212 \group_begin:
5213 \@@_open_shorten:
5214 \int_if_zero:nTF { #2 }
5215 { \color { nicematrix-first-col } }
5216 {
5217     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5218     { \color { nicematrix-last-col } }
5219 }
5220 \keys_set:nn { NiceMatrix / xdots } { #4 }
5221 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5222 \@@_actually_draw_Vdots:
5223 \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).

```

5224     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5225     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5226 }

```

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

```

5227 \NewDocumentCommand \@@_rotate: { 0 { } }
5228 {
5229     \peek_remove_spaces:n
5230     {
5231         \bool_gset_true:N \g_@@_rotate_bool
5232         \keys_set:nn { NiceMatrix / rotate } { #1 }
5233     }
5234 }
5235 \keys_define:nn { NiceMatrix / rotate }
5236 {
5237     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5238     c .value_forbidden:n = true ,
5239     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5240 }

```

## 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>

```

5241 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5242 {
5243   \tl_if_empty:nTF { #2 }
5244     { #1 }
5245     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5246 }
5247 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5248 { \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).

```

5249 \hook_gput_code:nnn { begindocument } { . }
5250 {
5251   \cs_set_nopar:Npn \l_@@_argspec_tl
5252     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5253   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5254   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5255     {
5256       \group_begin:
5257       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5258       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5259       \use:e
5260       {
5261         \@@_line_i:nn
5262           { \@@_double_int_eval:n #2 - \q_stop }
5263           { \@@_double_int_eval:n #3 - \q_stop }
5264       }
5265       \group_end:
5266     }
5267 }
5268 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5269 {
5270   \bool_set_false:N \l_@@_initial_open_bool
5271   \bool_set_false:N \l_@@_final_open_bool
5272   \bool_lazy_or:nnTF
5273     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5274     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5275     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

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

```

5276   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5277 }
5278 \hook_gput_code:nnn { begindocument } { . }
5279 {
5280   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5281     {
```

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_iii:nn`.

```

5282   \c_@@_pgfortikzpicture_tl
5283   \@@_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`.

```

5284         \c_@@_endpgfortikzpicture_tl
5285     }
5286 }

```

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

```

5287 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5288 {
5289     \pgfrememberpicturepositiononpagetrue
5290     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5291     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5292     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5293     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5294     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5295     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5296     \@@_draw_line:
5297 }

```

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`

```

5298 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5299 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

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

```

5300 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5301 {
5302     \tl_gput_right:Nx \g_@@_row_style_tl
5303     {

```

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

```

5304         \exp_not:N
5305         \@@_if_row_less_than:nn
5306         { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }

```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```

5307         { \exp_not:n { #1 } \scan_stop: }
5308     }
5309 }
5310 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5311 \keys_define:nn { NiceMatrix / RowStyle }
5312 {
5313   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5314   cell-space-top-limit .value_required:n = true ,
5315   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5316   cell-space-bottom-limit .value_required:n = true ,
5317   cell-space-limits .meta:n =
5318   {
5319     cell-space-top-limit = #1 ,
5320     cell-space-bottom-limit = #1 ,
5321   } ,
5322   color .tl_set:N = \l_@@_color_tl ,
5323   color .value_required:n = true ,
5324   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5325   bold .default:n = true ,
5326   nb-rows .code:n =
5327     \str_if_eq:nnTF { #1 } { * }
5328     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5329     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5330   nb-rows .value_required:n = true ,
5331   rowcolor .tl_set:N = \l_tmpa_tl ,
5332   rowcolor .value_required:n = true ,
5333   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5334 }

```

```

5335 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5336 {
5337   \group_begin:
5338   \tl_clear:N \l_tmpa_tl
5339   \tl_clear:N \l_@@_color_tl
5340   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5341   \dim_zero:N \l_tmpa_dim
5342   \dim_zero:N \l_tmpb_dim
5343   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5344   \tl_if_empty:NF \l_tmpa_tl
5345   {

```

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

```

5346     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5347     {

```

The command \@@\_exp\_color\_arg:No is *fully expandable*.

```

5348         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5349         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5350         { \int_use:N \c@iRow - * }
5351     }

```

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

```

5352     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5353     {
5354       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5355       {
5356         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5357         {
5358           \int_eval:n { \c@iRow + 1 }
5359           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5360         }
5361       }
5362     }
5363   }
5364   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

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

```

5365 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5366 {
5367   \exp_args:Nx \@@_put_in_row_style:n
5368   {
5369     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5370     {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5371       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5372       { \dim_use:N \l_tmpa_dim }
5373     }
5374   }
5375 }

```

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

```

5376 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5377 {
5378   \exp_args:Nx \@@_put_in_row_style:n
5379   {
5380     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5381     {
5382       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5383       { \dim_use:N \l_tmpb_dim }
5384     }
5385   }
5386 }

```

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

```

5387 \tl_if_empty:NF \l_@@_color_tl
5388 {
5389   \@@_put_in_row_style:e
5390   {
5391     \mode_leave_vertical:
5392     \@@_color:n { \l_@@_color_tl }
5393   }
5394 }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5395 \bool_if:NT \l_@@_bold_row_style_bool
5396 {
5397   \@@_put_in_row_style:n
5398   {
5399     \exp_not:n
5400     {
5401       \if_mode_math:
5402         \c_math_toggle_token
5403         \bfseries \boldmath
5404         \c_math_toggle_token
5405       \else:
5406         \bfseries \boldmath
5407       \fi:
5408     }
5409   }
5410 }
5411 \group_end:
5412 \g_@@_row_style_tl
5413 \ignorespaces
5414 }

```



## 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).

```
5415 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5416 {
```

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.

```
5417 \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`.

```
5418 \str_if_in:nnF { #1 } { !! }
5419 {
5420 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5421 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5422 }
5423 \int_if_zero:nTF \l_tmpa_int
```

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

```
5424 {
5425 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5426 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5427 }
```

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

```
5428 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5429 }

5430 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5431 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5432 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5433 {
5434 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5435 {
```

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

```

5436     \group_begin:
5437     \pgfsetcornersarced
5438     {
5439         \pgfpoint
5440         { \l_@@_tab_rounded_corners_dim }
5441         { \l_@@_tab_rounded_corners_dim }
5442     }

```

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.

```

5443     \bool_if:NTF \l_@@_hvlines_bool
5444     {
5445         \pgfpathrectanglecorners
5446         {
5447             \pgfpointadd
5448             { \@@_qpoint:n { row-1 } }
5449             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5450         }
5451         {
5452             \pgfpointadd
5453             {
5454                 \@@_qpoint:n
5455                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5456             }
5457             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5458         }
5459     }
5460     {
5461         \pgfpathrectanglecorners
5462         { \@@_qpoint:n { row-1 } }
5463         {
5464             \pgfpointadd
5465             {
5466                 \@@_qpoint:n
5467                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5468             }
5469             { \pgfpoint \c_zero_dim \arrayrulewidth }
5470         }
5471     }
5472     \pgfusepath { clip }
5473     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5474     }
5475 }

```

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`).

```

5476 \cs_new_protected:Npn \@@_actually_color:
5477 {
5478     \pgfpicture
5479     \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.

```

5480     \@@_clip_with_rounded_corners:
5481     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5482     {
5483         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5484     {
5485         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5486         \use:c { g_@@_color _ 1 _tl }
5487         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5488     }
5489     {
5490         \begin { pgfscope }
5491             \@@_color_opacity ##2
5492             \use:c { g_@@_color _ ##1 _tl }
5493             \tl_gclear:c { g_@@_color _ ##1 _tl }
5494             \pgfusepath { fill }
5495         \end { pgfscope }
5496     }
5497 }
5498 \endpgfpicture
5499 }

```

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

```

5500 \cs_new_protected:Npn \@@_color_opacity
5501 {
5502     \peek_meaning:NTF [
5503         { \@@_color_opacity:w }
5504         { \@@_color_opacity:w [ ] }
5505     }

```

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 currying.

```

5506 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5507 {
5508     \tl_clear:N \l_tmpa_tl
5509     \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.

```

5510     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5511     \tl_if_empty:NTF \l_tmpb_tl
5512         { \@declaredcolor }
5513         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5514 }

```

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

```

5515 \keys_define:nn { nicematrix / color-opacity }
5516 {
5517     opacity .tl_set:N          = \l_tmpa_tl ,
5518     opacity .value_required:n = true
5519 }

5520 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5521 {
5522     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5523     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5524     \@@_cartesian_path:
5525 }

```

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

```

5526 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5527 {
5528     \tl_if_blank:nF { #2 }
5529     {

```

```

5530 \@@_add_to_colors_seq:en
5531 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5532 { \@@_cartesian_color:nn { #3 } { - } }
5533 }
5534 }

```

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

```

5535 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5536 {
5537   \tl_if_blank:nF { #2 }
5538   {
5539     \@@_add_to_colors_seq:en
5540     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5541     { \@@_cartesian_color:nn { - } { #3 } }
5542   }
5543 }

```

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

```

5544 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5545 {
5546   \tl_if_blank:nF { #2 }
5547   {
5548     \@@_add_to_colors_seq:en
5549     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5550     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5551   }
5552 }

```

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

```

5553 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5554 {
5555   \tl_if_blank:nF { #2 }
5556   {
5557     \@@_add_to_colors_seq:en
5558     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5559     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5560   }
5561 }

```

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

```

5562 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5563 {
5564   \@@_cut_on_hyphen:w #1 \q_stop
5565   \tl_clear_new:N \l_@@_tmpc_tl
5566   \tl_clear_new:N \l_@@_tmpd_tl
5567   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5568   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5569   \@@_cut_on_hyphen:w #2 \q_stop
5570   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5571   \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`.

```

5572 \@@_cartesian_path:n { #3 }
5573 }

```

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

```

5574 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5575 {
5576   \clist_map_inline:nn { #3 }
5577   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5578 }

```

```

5579 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5580 {
5581   \int_step_inline:nn \c@iRow
5582   {
5583     \int_step_inline:nn \c@jCol
5584     {
5585       \int_if_even:nTF { #####1 + ##1 }
5586       { \@@_cellcolor [ #1 ] { #2 } }
5587       { \@@_cellcolor [ #1 ] { #3 } }
5588       { ##1 - #####1 }
5589     }
5590   }
5591 }

```

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”.

```

5592 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5593 {
5594   \@@_rectanglecolor [ #1 ] { #2 }
5595   { 1 - 1 }
5596   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5597 }

```

```

5598 \keys_define:nn { NiceMatrix / rowcolors }
5599 {
5600   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5601   respect-blocks .default:n = true ,
5602   cols .tl_set:N = \l_@@_cols_tl ,
5603   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5604   restart .default:n = true ,
5605   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5606 }

```

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*.

```

5607 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5608 {

```

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

```

5609   \group_begin:
5610   \seq_clear_new:N \l_@@_colors_seq
5611   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5612   \tl_clear_new:N \l_@@_cols_tl
5613   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5614   \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).

```

5615   \int_zero_new:N \l_@@_color_int
5616   \int_set_eq:NN \l_@@_color_int \c_one_int
5617   \bool_if:NT \l_@@_respect_blocks_bool
5618   {

```

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`).

```

5619     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5620     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5621     { \@@_not_in_exterior_p:nnnnn ##1 }
5622   }
5623   \pgfpicture
5624   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5625   \clist_map_inline:nn { #2 }
5626   {
5627     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5628     \tl_if_in:NnTF \l_tmpa_tl { - }
5629     { \@@_cut_on_hyphen:w ##1 \q_stop }
5630     { \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.

```

5631     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5632     \int_set:Nn \l_@@_color_int
5633     { \bool_if:NnTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5634     \int_zero_new:N \l_@@_tmpc_int
5635     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5636     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5637     {

```

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

```

5638         \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).

```

5639         \bool_if:NT \l_@@_respect_blocks_bool
5640         {
5641           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5642           { \@@_intersect_our_row_p:nnnnn #####1 }
5643           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5644         }
5645         \tl_set:No \l_@@_rows_tl
5646         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5647         \tl_clear_new:N \l_@@_color_tl
5648         \tl_set:Nx \l_@@_color_tl
5649         {
5650           \@@_color_index:n
5651           {
5652             \int_mod:nn
5653             { \l_@@_color_int - 1 }
5654             { \seq_count:N \l_@@_colors_seq }
5655             + 1
5656           }
5657         }
5658         \tl_if_empty:NF \l_@@_color_tl
5659         {
5660           \@@_add_to_colors_seq:ee
5661           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5662           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5663         }
5664         \int_incr:N \l_@@_color_int
5665         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5666       }
5667     }
5668   \endpgfpicture

```

```

5669 \group_end:
5670 }

```

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.

```

5671 \cs_new:Npn \@@_color_index:n #1
5672 {
5673   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5674   { \@@_color_index:n { #1 - 1 } }
5675   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5676 }

```

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

```

5677 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5678 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5679 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5680 {
5681   \int_compare:nNnT { #3 } > \l_tmpb_int
5682   { \int_set:Nn \l_tmpb_int { #3 } }
5683 }

```

```

5684 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5685 {
5686   \int_if_zero:nTF { #4 }
5687   \prg_return_false:
5688   {
5689     \int_compare:nNnTF { #2 } > \c@jCol
5690     \prg_return_false:
5691     \prg_return_true:
5692   }
5693 }

```

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

```

5694 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5695 {
5696   \int_compare:nNnTF { #1 } > \l_tmpa_int
5697   \prg_return_false:
5698   {
5699     \int_compare:nNnTF \l_tmpa_int > { #3 }
5700     \prg_return_false:
5701     \prg_return_true:
5702   }
5703 }

```

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`).

```

5704 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5705 {
5706   \dim_compare:nNnTF { #1 } = \c_zero_dim
5707   {
5708     \bool_if:NTF

```

```

5709     \l_@@_nocolor_used_bool
5710     \@@_cartesian_path_normal_ii:
5711     {
5712         \seq_if_empty:NTF \l_@@_corners_cells_seq
5713         { \@@_cartesian_path_normal_i:n { #1 } }
5714         \@@_cartesian_path_normal_ii:
5715     }
5716 }
5717 { \@@_cartesian_path_normal_i:n { #1 } }
5718 }

```

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.

```

5719 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5720 {
5721     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5722     \clist_map_inline:Nn \l_@@_cols_tl
5723     {
5724         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5725         \tl_if_in:NnTF \l_tmpa_tl { - }
5726         { \@@_cut_on_hyphen:w ##1 \q_stop }
5727         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5728         \tl_if_empty:NTF \l_tmpa_tl
5729         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5730         {
5731             \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5732             { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5733         }
5734         \tl_if_empty:NTF \l_tmpb_tl
5735         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5736         {
5737             \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5738             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5739         }
5740         \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5741         { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

\l\_@@\_tmpc\_tl will contain the number of column.

```

5742         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5743         \@@_qpoint:n { col - \l_tmpa_tl }
5744         \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5745         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5746         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5747         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5748         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5749     \clist_map_inline:Nn \l_@@_rows_tl
5750     {
5751         \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5752         \tl_if_in:NnTF \l_tmpa_tl { - }
5753         { \@@_cut_on_hyphen:w #####1 \q_stop }
5754         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5755         \tl_if_empty:NTF \l_tmpa_tl
5756         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5757         {
5758             \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5759             { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5760         }
5761         \tl_if_empty:NTF \l_tmpb_tl
5762         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5763         {

```



```

5764         \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5765         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5766     }
5767     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5768     { \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`.

```

5769     \cs_if_exist:cF
5770     { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5771     {
5772         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5773         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5774         \@@_qpoint:n { row - \l_tmpa_tl }
5775         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5776         \pgfpathrectanglecorners
5777         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5778         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5779     }
5780 }
5781 }
5782 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5783 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5784 {
5785     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5786     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5787     \clist_map_inline:Nn \l_@@_cols_tl
5788     {
5789         \@@_qpoint:n { col - ##1 }
5790         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5791         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5792         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5793         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5794         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5795     \clist_map_inline:Nn \l_@@_rows_tl
5796     {
5797         \seq_if_in:NnF \l_@@_corners_cells_seq
5798         { #####1 - ##1 }
5799         {
5800             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5801             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5802             \@@_qpoint:n { row - #####1 }
5803             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5804             \cs_if_exist:cF { @@ _ #####1 _ ##1 _ nocolor }
5805             {
5806                 \pgfpathrectanglecorners
5807                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5808                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5809             }
5810         }
5811     }
5812 }
5813 }

```

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`).

```

5814 \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

```

5815 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5816 {
5817   \bool_set_true:N \l_@@_nocolor_used_bool
5818   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5819   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5820   \clist_map_inline:Nn \l_@@_rows_tl
5821   {
5822     \clist_map_inline:Nn \l_@@_cols_tl
5823     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5824   }
5825 }

```

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`.

```

5826 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5827 {
5828   \clist_set_eq:NN \l_tmpa_clist #1
5829   \clist_clear:N #1
5830   \clist_map_inline:Nn \l_tmpa_clist
5831   {
5832     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5833     \tl_if_in:NnTF \l_tmpa_tl { - }
5834     { \@@_cut_on_hyphen:w ##1 \q_stop }
5835     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5836     \bool_lazy_or:nnT
5837     { \tl_if_blank_p:o \l_tmpa_tl }
5838     { \str_if_eq_p:on \l_tmpa_tl { * } }
5839     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5840     \bool_lazy_or:nnT
5841     { \tl_if_blank_p:o \l_tmpb_tl }
5842     { \str_if_eq_p:on \l_tmpb_tl { * } }
5843     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5844     \int_compare:nNnT \l_tmpb_tl > #2
5845     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5846     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5847     { \clist_put_right:Nn #1 { #####1 } }
5848   }
5849 }

```

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

```

5850 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5851 {
5852   \@@_test_color_inside:
5853   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5854   {

```

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`).

```

5855     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5856     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5857   }
5858   \ignorespaces
5859 }

```

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

```

5860 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5861 {
5862   \@@_test_color_inside:
5863   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5864   {
5865     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5866     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5867     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5868   }
5869   \ignorespaces
5870 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5871 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5872 { \@@_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.

```

5873 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5874 {
5875   \@@_test_color_inside:
5876   \peek_remove_spaces:n
5877   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5878 }

5879 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5880 {

```

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`.

```

5881 \seq_gclear:N \g_tmpa_seq
5882 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5883 { \@@_rowlistcolors_tabular_i:nnnn #1 }
5884 \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`.

```

5885 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5886 {
5887   { \int_use:N \c@iRow }
5888   { \exp_not:n { #1 } }
5889   { \exp_not:n { #2 } }
5890   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5891 }
5892 }

```

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`).

```
5893 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5894 {
5895   \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).

```
5896   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5897   {
5898     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5899     {
5900       \@@_rowlistcolors
5901       [ \exp_not:n { #2 } ]
5902       { #1 - \int_eval:n { \c@iRow - 1 } }
5903       { \exp_not:n { #3 } }
5904       [ \exp_not:n { #4 } ]
5905     }
5906   }
5907 }
```

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.

```
5908 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5909 {
5910   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5911   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5912   \seq_gclear:N \g_@@_rowlistcolors_seq
5913 }

5914 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5915 {
5916   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5917   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5918 }
```

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.

```
5919 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5920 {
```

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).

```
5921   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5922   {
```

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).

```
5923   \tl_gput_left:Nx \g_@@_pre_code_before_tl
5924   {
5925     \exp_not:N \columncolor [ #1 ]
5926     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5927   }
5928 }
5929 }
```

```

5930 \hook_gput_code:nnn { begindocument } { . }
5931 {
5932   \IfPackageLoadedTF { colortbl }
5933   {
5934     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5935     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5936     \cs_new_protected:Npn \@@_revert_colortbl:
5937     {
5938       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5939       {
5940         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5941         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5942       }
5943     }
5944   }
5945   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5946 }

```

## 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`.

```

5947 \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`.

```

5948 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5949 {
5950   \int_if_zero:nTF \l_@@_first_col_int
5951   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5952   {
5953     \int_if_zero:nTF \c@jCol
5954     {
5955       \int_compare:nNf \c@iRow = { -1 }
5956       { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5957     }
5958     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5959   }
5960 }

```

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.

```

5961 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5962 {
5963   \int_if_zero:nF \c@iRow
5964   {
5965     \int_compare:nNf \c@iRow = \l_@@_last_row_int
5966     {

```

```

5967         \int_compare:nNtT \c@jCol > \c_zero_int
5968         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5969     }
5970 }
5971 }

```

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:nNtT \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.

```

5972 \keys_define:nn { NiceMatrix / Rules }
5973 {
5974     position .int_set:N = \l_@@_position_int ,
5975     position .value_required:n = true ,
5976     start .int_set:N = \l_@@_start_int ,
5977     end .code:n =
5978         \bool_lazy_or:nnTF
5979         { \tl_if_empty_p:n { #1 } }
5980         { \str_if_eq_p:nn { #1 } { last } }
5981         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5982         { \int_set:Nn \l_@@_end_int { #1 } }
5983 }

```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

5984 \keys_define:nn { NiceMatrix / RulesBis }
5985 {
5986     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5987     multiplicity .initial:n = 1 ,
5988     dotted .bool_set:N = \l_@@_dotted_bool ,
5989     dotted .initial:n = false ,
5990     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]`.

```

5991     color .code:n =
5992         \@@_set_CT@arc@:n { #1 }
5993     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5994     color .value_required:n = true ,
5995     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5996     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.

```

5997     tikz .code:n =
5998         \IfPackageLoadedTF { tikz }
5999         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6000         { \@@_error:n { tikz~without~tikz } } ,
6001     tikz .value_required:n = true ,
6002     total-width .dim_set:N = \l_@@_rule_width_dim ,

```

```

6003     total-width .value_required:n = true ,
6004     width .meta:n = { total-width = #1 } ,
6005     unknown .code:n = \@_error:n { Unknow~key~for~RulesBis }
6006 }

```

## The vertical rules

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

```

6007 \cs_new_protected:Npn \@_vline:n #1
6008 {

```

The group is for the options.

```

6009     \group_begin:
6010     \int_set_eq:NN \l_@@_end_int \c@iRow
6011     \keys_set_known:nN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6012     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6013     \@_vline_i:
6014     \group_end:
6015 }

```

```

6016 \cs_new_protected:Npn \@_vline_i:
6017 {

```

`\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`.

```

6018     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6019     \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6020     \l_tmpa_tl
6021     {

```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small vertical rule won't be drawn.

```

6022         \bool_gset_true:N \g_tmpa_bool
6023         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6024         { \@_test_vline_in_block:nnnnn ##1 }
6025         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6026         { \@_test_vline_in_block:nnnnn ##1 }
6027         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6028         { \@_test_vline_in_stroken_block:nnnn ##1 }
6029         \clist_if_empty:NF \l_@@_corners_clist \@_test_in_corner_v:
6030         \bool_if:NTF \g_tmpa_bool
6031         {
6032             \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.

```

6033             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6034         }
6035         {
6036             \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6037             {
6038                 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6039                 \@_vline_ii:
6040                 \int_zero:N \l_@@_local_start_int
6041             }
6042         }
6043     }
6044     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6045     {

```

```

6046         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6047         \@@_vline_ii:
6048     }
6049 }

6050 \cs_new_protected:Npn \@@_test_in_corner_v:
6051 {
6052     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6053     {
6054         \seq_if_in:NxT
6055         \l_@@_corners_cells_seq
6056         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6057         { \bool_set_false:N \g_tmpa_bool }
6058     }
6059     {
6060         \seq_if_in:NxT
6061         \l_@@_corners_cells_seq
6062         { \l_tmpa_tl - \l_tmpb_tl }
6063         {
6064             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6065             { \bool_set_false:N \g_tmpa_bool }
6066             {
6067                 \seq_if_in:NxT
6068                 \l_@@_corners_cells_seq
6069                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6070                 { \bool_set_false:N \g_tmpa_bool }
6071             }
6072         }
6073     }
6074 }

6075 \cs_new_protected:Npn \@@_vline_ii:
6076 {
6077     \tl_clear:N \l_@@_tikz_rule_tl
6078     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6079     \bool_if:NTF \l_@@_dotted_bool
6080     \@@_vline_iv:
6081     {
6082         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6083         \@@_vline_iii:
6084         \@@_vline_v:
6085     }
6086 }

```

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

```

6087 \cs_new_protected:Npn \@@_vline_iii:
6088 {
6089     \pgfpicture
6090     \pgfrememberpicturepositiononpagetrue
6091     \pgf@relevantforpicturesizefalse
6092     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6093     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6094     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6095     \dim_set:Nn \l_tmpb_dim
6096     {
6097         \pgf@x
6098         - 0.5 \l_@@_rule_width_dim
6099         +
6100         ( \arrayrulewidth * \l_@@_multiplicity_int
6101           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6102     }

```



```

6103 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6104 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6105 \bool_lazy_all:nT
6106 {
6107   { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6108   { \cs_if_exist_p:N \CT@drsc@ }
6109   { ! \tl_if_blank_p:o \CT@drsc@ }
6110 }
6111 {
6112   \group_begin:
6113   \CT@drsc@
6114   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6115   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6116   \dim_set:Nn \l_@@_tmpd_dim
6117   {
6118     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6119     * ( \l_@@_multiplicity_int - 1 )
6120   }
6121   \pgfpathrectanglecorners
6122   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6123   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6124   \pgfusepath { fill }
6125   \group_end:
6126 }
6127 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6128 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6129 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6130 {
6131   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6132   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6133   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6134   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6135 }
6136 \CT@arc@
6137 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6138 \pgfsetrectcap
6139 \pgfusepathqstroke
6140 \endpgfpicture
6141 }

```

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

```

6142 \cs_new_protected:Npn \@@_vline_iv:
6143 {
6144   \pgfpicture
6145   \pgfrememberpicturepositiononpagetrue
6146   \pgf@relevantforpicturesizefalse
6147   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6148   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6149   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6150   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6151   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6152   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6153   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6154   \CT@arc@
6155   \@@_draw_line:
6156   \endpgfpicture
6157 }

```

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

```

6158 \cs_new_protected:Npn \@@_vline_v:
6159 {
6160   \begin {tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6161 \CT@arc@
6162 \tl_if_empty:NF \l_@@_rule_color_tl
6163 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6164 \pgfrememberpicturepositiononpagetrue
6165 \pgf@relevantforpicturesizefalse
6166 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6167 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6168 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6169 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6170 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6171 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6172 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6173 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6174 ( \l_tmpb_dim , \l_tmpa_dim ) --
6175 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6176 \end { tikzpicture }
6177 }

```

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).

```

6178 \cs_new_protected:Npn \@@_draw_vlines:
6179 {
6180   \int_step_inline:nnn
6181   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6182   {
6183     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6184     \c@jCol
6185     { \int_eval:n { \c@jCol + 1 } }
6186   }
6187   {
6188     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6189     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6190     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6191   }
6192 }

```

## 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}`.

```

6193 \cs_new_protected:Npn \@@_hline:n #1
6194 {

```

The group is for the options.

```

6195   \group_begin:
6196   \int_zero_new:N \l_@@_end_int
6197   \int_set_eq:NN \l_@@_end_int \c@jCol
6198   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6199   \@@_hline_i:
6200   \group_end:
6201 }
6202 \cs_new_protected:Npn \@@_hline_i:
6203 {
6204   \int_zero_new:N \l_@@_local_start_int
6205   \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.

```

6206     \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6207     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6208         \l_tmpb_tl
6209     {

```

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.

```

6210         \bool_gset_true:N \g_tmpa_bool
6211         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6212             { \@@_test_hline_in_block:nnnnn ##1 }
6213         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6214             { \@@_test_hline_in_block:nnnnn ##1 }
6215         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6216             { \@@_test_hline_in_stroken_block:nnnn ##1 }
6217         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6218         \bool_if:NTF \g_tmpa_bool
6219             {
6220                 \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.

```

6221             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6222         }
6223     {
6224         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6225         {
6226             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6227             \@@_hline_ii:
6228             \int_zero:N \l_@@_local_start_int
6229         }
6230     }
6231 }
6232 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6233 {
6234     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6235     \@@_hline_ii:
6236 }
6237 }

```

```

6238 \cs_new_protected:Npn \@@_test_in_corner_h:
6239 {
6240     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6241     {
6242         \seq_if_in:NxT
6243             \l_@@_corners_cells_seq
6244             { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6245             { \bool_set_false:N \g_tmpa_bool }
6246     }
6247     {
6248         \seq_if_in:NxT
6249             \l_@@_corners_cells_seq
6250             { \l_tmpa_tl - \l_tmpb_tl }
6251         {
6252             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6253             { \bool_set_false:N \g_tmpa_bool }
6254             {
6255                 \seq_if_in:NxT
6256                     \l_@@_corners_cells_seq
6257                     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }

```

```

6258             { \bool_set_false:N \g_tmpa_bool }
6259         }
6260     }
6261 }
6262 }

6263 \cs_new_protected:Npn \@@_hline_ii:
6264 {
6265     \tl_clear:N \l_@@_tikz_rule_tl
6266     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6267     \bool_if:NTF \l_@@_dotted_bool
6268         \@@_hline_iv:
6269     {
6270         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6271             \@@_hline_iii:
6272             \@@_hline_v:
6273     }
6274 }

```

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

```

6275 \cs_new_protected:Npn \@@_hline_iii:
6276 {
6277     \pgfpicture
6278     \pgfrememberpicturepositiononpagetrue
6279     \pgf@relevantforpicturesizefalse
6280     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6281     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6282     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6283     \dim_set:Nn \l_tmpb_dim
6284     {
6285         \pgf@y
6286         - 0.5 \l_@@_rule_width_dim
6287         +
6288         ( \arrayrulewidth * \l_@@_multiplicity_int
6289           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6290     }
6291     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6292     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6293     \bool_lazy_all:nT
6294     {
6295         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6296         { \cs_if_exist_p:N \CT@drsc@ }
6297         { ! \tl_if_blank_p:o \CT@drsc@ }
6298     }
6299     {
6300         \group_begin:
6301         \CT@drsc@
6302         \dim_set:Nn \l_@@_tmpd_dim
6303         {
6304             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6305             * ( \l_@@_multiplicity_int - 1 )
6306         }
6307         \pgfpathrectanglecorners
6308         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6309         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6310         \pgfusepathqfill
6311         \group_end:
6312     }
6313     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6314     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6315     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6316     {

```

```

6317     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6318     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6319     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6320     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6321   }
6322   \CT@arc@
6323   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6324   \pgfsetrectcap
6325   \pgfusepathqstroke
6326   \endpgfpicture
6327 }

```

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}

```

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

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}

```

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

```

6328 \cs_new_protected:Npn \@@_hline_iv:
6329 {
6330   \pgfpicture
6331   \pgfrememberpicturepositiononpagetrue
6332   \pgf@relevantforpicturesizefalse
6333   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6334   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6335   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6336   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6337   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6338   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6339   {
6340     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6341     \bool_if:NF \g_@@_delims_bool
6342     { \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.

```

6343     \tl_if_eq:NnF \g_@@_left_delim_tl (
6344       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6345     )
6346     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6347     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6348     \int_compare:nNnT \l_@@_local_end_int = \c_j_col
6349     {
6350       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6351       \bool_if:NF \g_@@_delims_bool
6352       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6353       \tl_if_eq:NnF \g_@@_right_delim_tl (
6354         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6355       )
6356     }
6357     \CT@arc@
6358     \@@_draw_line:

```

```

6358 \endpgfpicture
6359 }

```

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`).

```

6360 \cs_new_protected:Npn \@@_hline_v:
6361 {
6362   \begin { tikzpicture }

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6363   \CT@arc@
6364   \tl_if_empty:NF \l_@@_rule_color_tl
6365   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6366   \pgfrememberpicturepositiononpagetrue
6367   \pgf@relevantforpicturesizefalse
6368   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6369   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6370   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6371   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6372   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6373   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6374   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6375   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6376   ( \l_tmpa_dim , \l_tmpb_dim ) --
6377   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6378   \end { tikzpicture }
6379 }

```

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).

```

6380 \cs_new_protected:Npn \@@_draw_hlines:
6381 {
6382   \int_step_inline:nnn
6383   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6384   {
6385     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6386     \c@iRow
6387     { \int_eval:n { \c@iRow + 1 } }
6388   }
6389   {
6390     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6391     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6392     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6393   }
6394 }

```

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

```

6395 \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.

```

6396 \cs_set:Npn \@@_Hline_i:n #1
6397 {
6398   \peek_remove_spaces:n
6399   {
6400     \peek_meaning:NNTF \Hline
6401     { \@@_Hline_ii:nn { #1 + 1 } }
6402     { \@@_Hline_iii:n { #1 } }
6403   }
6404 }

```

```

6405 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6406 \cs_set:Npn \@@_Hline_iii:n #1
6407 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6408 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6409 {
6410   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6411   \skip_vertical:N \l_@@_rule_width_dim
6412   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6413   {
6414     \@@_hline:n
6415     {
6416       multiplicity = #1 ,
6417       position = \int_eval:n { \c@iRow + 1 } ,
6418       total-width = \dim_use:N \l_@@_rule_width_dim ,
6419       #2
6420     }
6421   }
6422   \egroup
6423 }

```

### 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`).

```

6424 \cs_new_protected:Npn \@@_custom_line:n #1
6425 {
6426   \str_clear_new:N \l_@@_command_str
6427   \str_clear_new:N \l_@@_ccommand_str
6428   \str_clear_new:N \l_@@_letter_str
6429   \tl_clear_new:N \l_@@_other_keys_tl
6430   \keys_set:known { 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.

```

6431   \bool_lazy_all:nTF
6432   {
6433     { \str_if_empty_p:N \l_@@_letter_str }
6434     { \str_if_empty_p:N \l_@@_command_str }
6435     { \str_if_empty_p:N \l_@@_ccommand_str }
6436   }
6437   { \@@_error:n { No~letter~and~no~command } }
6438   { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6439 }
6440 \keys_define:nn { NiceMatrix / custom-line }
6441 {
6442   letter .str_set:N = \l_@@_letter_str ,
6443   letter .value_required:n = true ,
6444   command .str_set:N = \l_@@_command_str ,
6445   command .value_required:n = true ,
6446   ccommand .str_set:N = \l_@@_ccommand_str ,
6447   ccommand .value_required:n = true ,
6448 }
6449 \cs_new_protected:Npn \@@_custom_line_i:n #1
6450 {

```

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

```

6451 \bool_set_false:N \l_@@_tikz_rule_bool
6452 \bool_set_false:N \l_@@_dotted_rule_bool
6453 \bool_set_false:N \l_@@_color_bool
6454 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6455 \bool_if:NT \l_@@_tikz_rule_bool
6456 {
6457   \IfPackageLoadedTF { tikz }
6458   { }
6459   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6460   \bool_if:NT \l_@@_color_bool
6461   { \@@_error:n { color-in-custom-line-with-tikz } }
6462 }
6463 \bool_if:NT \l_@@_dotted_rule_bool
6464 {
6465   \int_compare:nNt \l_@@_multiplicity_int > \c_one_int
6466   { \@@_error:n { key-multiplicity-with-dotted } }
6467 }
6468 \str_if_empty:NF \l_@@_letter_str
6469 {
6470   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6471   { \@@_error:n { Several~letters } }
6472   {
6473     \exp_args:NnV \tl_if_in:NnTF
6474     \c_@@_forbidden_letters_str \l_@@_letter_str
6475     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6476     {

```

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.

```

6477 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6478 { \@@_v_custom_line:n { #1 } }
6479 }
6480 }
6481 }
6482 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6483 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6484 }
6485 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6486 \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.

```

6487 \keys_define:nn { NiceMatrix / custom-line-bis }
6488 {
6489   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6490   multiplicity .initial:n = 1 ,
6491   multiplicity .value_required:n = true ,
6492   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6493   color .value_required:n = true ,
6494   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6495   tikz .value_required:n = true ,
6496   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6497   dotted .value_forbidden:n = true ,
6498   total-width .code:n = { } ,
6499   total-width .value_required:n = true ,
6500   width .code:n = { } ,
6501   width .value_required:n = true ,

```



```

6502     sep-color .code:n = { } ,
6503     sep-color .value_required:n = true ,
6504     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6505 }

```

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

```

6506 \bool_new:N \l_@@_dotted_rule_bool
6507 \bool_new:N \l_@@_tikz_rule_bool
6508 \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`.

```

6509 \keys_define:nn { NiceMatrix / custom-line-width }
6510 {
6511     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6512     multiplicity .initial:n = 1 ,
6513     multiplicity .value_required:n = true ,
6514     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6515     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6516                     \bool_set_true:N \l_@@_total_width_bool ,
6517     total-width .value_required:n = true ,
6518     width .meta:n = { total-width = #1 } ,
6519     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6520 }

```

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`).

```

6521 \cs_new_protected:Npn \@@_h_custom_line:n #1
6522 {

```

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`).

```

6523     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6524     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6525 }

```

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`).

```

6526 \cs_new_protected:Npn \@@_c_custom_line:n #1
6527 {

```

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

```

6528     \exp_args:Nc \NewExpandableDocumentCommand
6529     { nicematrix - \l_@@_ccommand_str }
6530     { 0 { } m }
6531     {
6532         \noalign
6533         {
6534             \@@_compute_rule_width:n { #1 , ##1 }
6535             \skip_vertical:n { \l_@@_rule_width_dim }
6536             \clist_map_inline:nn
6537             { ##2 }
6538             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6539         }
6540     }
6541     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6542 }

```

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*.

```

6543 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6544 {
6545   \str_if_in:nnTF { #2 } { - }
6546   { \@@_cut_on_hyphen:w #2 \q_stop }
6547   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6548   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6549   {
6550     \@@_hline:n
6551     {
6552       #1 ,
6553       start = \l_tmpa_tl ,
6554       end = \l_tmpb_tl ,
6555       position = \int_eval:n { \c@iRow + 1 } ,
6556       total-width = \dim_use:N \l_@@_rule_width_dim
6557     }
6558   }
6559 }

6560 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6561 {
6562   \bool_set_false:N \l_@@_tikz_rule_bool
6563   \bool_set_false:N \l_@@_total_width_bool
6564   \bool_set_false:N \l_@@_dotted_rule_bool
6565   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6566   \bool_if:NF \l_@@_total_width_bool
6567   {
6568     \bool_if:NTF \l_@@_dotted_rule_bool
6569     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6570     {
6571       \bool_if:NF \l_@@_tikz_rule_bool
6572       {
6573         \dim_set:Nn \l_@@_rule_width_dim
6574         {
6575           \arrayrulewidth * \l_@@_multiplicity_int
6576           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6577         }
6578       }
6579     }
6580   }
6581 }

6582 \cs_new_protected:Npn \@@_v_custom_line:n #1
6583 {
6584   \@@_compute_rule_width:n { #1 }

```

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

```

6585   \tl_gput_right:Nx \g_@@_array_preamble_tl
6586   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6587   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6588   {
6589     \@@_vline:n
6590     {
6591       #1 ,
6592       position = \int_eval:n { \c@jCol + 1 } ,
6593       total-width = \dim_use:N \l_@@_rule_width_dim
6594     }
6595   }
6596   \@@_rec_preamble:n
6597 }

6598 \@@_custom_line:n
6599 { 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.

```
6600 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6601 {
6602   \int_compare:nNnT \l_tmpa_tl > { #1 }
6603   {
6604     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6605     {
6606       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6607       {
6608         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6609         { \bool_gset_false:N \g_tmpa_bool }
6610       }
6611     }
6612   }
6613 }
```

The same for vertical rules.

```
6614 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6615 {
6616   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6617   {
6618     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6619     {
6620       \int_compare:nNnT \l_tmpb_tl > { #2 }
6621       {
6622         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6623         { \bool_gset_false:N \g_tmpa_bool }
6624       }
6625     }
6626   }
6627 }

6628 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6629 {
6630   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6631   {
6632     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6633     {
6634       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6635       { \bool_gset_false:N \g_tmpa_bool }
6636       {
6637         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6638         { \bool_gset_false:N \g_tmpa_bool }
6639       }
6640     }
6641   }
6642 }

6643 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6644 {
6645   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6646   {
6647     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6648     {
6649       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6650       { \bool_gset_false:N \g_tmpa_bool }
6651       {
6652         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6653         { \bool_gset_false:N \g_tmpa_bool }
6654       }
6655     }
6656   }
6657 }
```

```

6655     }
6656   }
6657 }

```

## 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.

```

6658 \cs_new_protected:Npn \@@_compute_corners:
6659 {

```

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.

```

6660   \seq_clear_new:N \l_@@_corners_cells_seq
6661   \clist_map_inline:Nn \l_@@_corners_clist
6662   {
6663     \str_case:nnF { ##1 }
6664     {
6665       { NW }
6666       { \@@_compute_a_corner:nnnnnn 1 1 1 \c@iRow \c@jCol }
6667       { NE }
6668       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6669       { SW }
6670       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6671       { SE }
6672       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6673     }
6674     { \@@_error:nn { bad~corner } { ##1 } }
6675   }

```

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

```

6676   \seq_if_empty:NF \l_@@_corners_cells_seq
6677   {

```

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.

```

6678     \tl_gput_right:Nx \g_@@_aux_tl
6679     {
6680       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6681       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6682     }
6683   }
6684 }

```

“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.

```

6685 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6686 {

```

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.

```

6687 \bool_set_false:N \l_tmpa_bool
6688 \int_zero_new:N \l_@@_last_empty_row_int
6689 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6690 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6691 {
6692   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6693   \bool_lazy_or:nnTF
6694   {
6695     \cs_if_exist_p:c
6696     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6697   }
6698   \l_tmpb_bool
6699   { \bool_set_true:N \l_tmpa_bool }
6700   {
6701     \bool_if:NF \l_tmpa_bool
6702     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6703   }
6704 }

```

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

```

6705 \bool_set_false:N \l_tmpa_bool
6706 \int_zero_new:N \l_@@_last_empty_column_int
6707 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6708 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6709 {
6710   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6711   \bool_lazy_or:nnTF
6712   \l_tmpb_bool
6713   {
6714     \cs_if_exist_p:c
6715     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6716   }
6717   { \bool_set_true:N \l_tmpa_bool }
6718   {
6719     \bool_if:NF \l_tmpa_bool
6720     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6721   }
6722 }

```

Now, we loop over the rows.

```

6723 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6724 {

```

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

```

6725 \bool_set_false:N \l_tmpa_bool
6726 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6727 {
6728   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6729   \bool_lazy_or:nnTF
6730   \l_tmpb_bool
6731   {
6732     \cs_if_exist_p:c
6733     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6734   }
6735   { \bool_set_true:N \l_tmpa_bool }
6736   {
6737     \bool_if:NF \l_tmpa_bool
6738     {
6739       \int_set:Nn \l_@@_last_empty_column_int { #####1 }

```

```

6740         \seq_put_right:Nn
6741         \l_@@_corners_cells_seq
6742         { ##1 - #####1 }
6743     }
6744 }
6745 }
6746 }
6747 }

```

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`).

```

6748 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6749 {
6750     \int_set:Nn \l_tmpa_int { #1 }
6751     \int_set:Nn \l_tmpb_int { #2 }
6752     \bool_set_false:N \l_tmpb_bool
6753     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6754     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6755 }
6756 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6757 {
6758     \int_compare:nNnF { #3 } > { #1 }
6759     {
6760         \int_compare:nNnF { #1 } > { #5 }
6761         {
6762             \int_compare:nNnF { #4 } > { #2 }
6763             {
6764                 \int_compare:nNnF { #2 } > { #6 }
6765                 { \bool_set_true:N \l_tmpb_bool }
6766             }
6767         }
6768     }
6769 }

```

## 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.

```

6770 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6771 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6772 {
6773     auto-columns-width .code:n =
6774     {
6775         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6776         \dim_gzero_new:N \g_@@_max_cell_width_dim
6777         \bool_set_true:N \l_@@_auto_columns_width_bool
6778     }
6779 }
6780 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6781 {
6782     \int_gincr:N \g_@@_NiceMatrixBlock_int
6783     \dim_zero:N \l_@@_columns_width_dim

```

```

6784 \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6785 \bool_if:NT \l_@@_block_auto_columns_width_bool
6786 {
6787   \cs_if_exist:cT
6788   { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6789   {
6790     % is \exp_args:Nne mandatory?
6791     \exp_args:Nne \dim_set:Nn \l_@@_columns_width_dim
6792     {
6793       \use:c
6794       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6795     }
6796   }
6797 }
6798 }

```

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`).

```

6799 {
6800   \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.

```

6801   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6802   {
6803     \bool_if:NT \l_@@_block_auto_columns_width_bool
6804     {
6805       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6806       \iow_shipout:Nx \@mainaux
6807       {
6808         \cs_gset:cpn
6809         { @@ _ 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.

```

6810         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6811       }
6812       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6813     }
6814   }
6815   \ignorespacesafterend
6816 }

```

## 26 The extra nodes

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

```

6817 \cs_generate_variant:Nn \dim_min:nn { v n }
6818 \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).

```

6819 \cs_new_protected:Npn \@@_create_extra_nodes:
6820 {
6821   \bool_if:nTF \l_@@_medium_nodes_bool
6822   {
6823     \bool_if:NTF \l_@@_large_nodes_bool
6824     \@@_create_medium_and_large_nodes:
6825     \@@_create_medium_nodes:
6826   }

```

```

6827     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6828 }

```

We have three macros of creation of nodes: `\@@_create_medium_nodes:`, `\@@_create_large_nodes:` and `\@@_create_medium_and_large_nodes:`.

We have to compute the mathematical coordinates of the “medium nodes”. These mathematical coordinates are also used to compute the mathematical coordinates of the “large nodes”. That’s why we write a command `\@@_computations_for_medium_nodes:` to do these computations.

The command `\@@_computations_for_medium_nodes:` must be used in a `{pgfpicture}`.

For each row  $i$ , we compute two dimensions `l_@@_row_i_min_dim` and `l_@@_row_i_max_dim`. The dimension `l_@@_row_i_min_dim` is the minimal  $y$ -value of all the cells of the row  $i$ . The dimension `l_@@_row_i_max_dim` is the maximal  $y$ -value of all the cells of the row  $i$ .

Similarly, for each column  $j$ , we compute two dimensions `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`. The dimension `l_@@_column_j_min_dim` is the minimal  $x$ -value of all the cells of the column  $j$ . The dimension `l_@@_column_j_max_dim` is the maximal  $x$ -value of all the cells of the column  $j$ .

Since these dimensions will be computed as maximum or minimum, we initialize them to `\c_max_dim` or `-\c_max_dim`.

```

6829 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6830 {
6831   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6832   {
6833     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6834     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6835     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6836     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6837   }
6838   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6839   {
6840     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6841     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6842     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6843     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6844   }

```

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

```

6845   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6846   {
6847     \int_step_variable:nnNn
6848     \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.

```

6849     {
6850       \cs_if_exist:cT
6851       { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

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

```

6852     {
6853       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6854       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6855       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6856       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6857       {
6858         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6859         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6860       }

```



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

```

6861         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6862         \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6863         { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6864         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6865         {
6866             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6867             { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6868         }
6869     }
6870 }
6871 }

```

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

```

6872     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6873     {
6874         \dim_compare:nNnT
6875         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6876         {
6877             \@@_qpoint:n { row - \@@_i: - base }
6878             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6879             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6880         }
6881     }
6882     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6883     {
6884         \dim_compare:nNnT
6885         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6886         {
6887             \@@_qpoint:n { col - \@@_j: }
6888             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6889             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6890         }
6891     }
6892 }

```

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

```

6893 \cs_new_protected:Npn \@@_create_medium_nodes:
6894 {
6895     \pgfpicture
6896     \pgfrememberpicturepositiononpagetrue
6897     \pgf@relevantforpicturesizefalse
6898     \@@_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”.

```

6899         \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6900         \@@_create_nodes:
6901     \endpgfpicture
6902 }

```

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:`.

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6903 \cs_new_protected:Npn \@@_create_large_nodes:
6904 {
6905   \pgfpicture
6906     \pgfrememberpicturepositiononpagetrue
6907     \pgf@relevantforpicturesizefalse
6908     \@@_computations_for_medium_nodes:
6909     \@@_computations_for_large_nodes:
6910     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6911     \@@_create_nodes:
6912   \endpgfpicture
6913 }
6914 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6915 {
6916   \pgfpicture
6917     \pgfrememberpicturepositiononpagetrue
6918     \pgf@relevantforpicturesizefalse
6919     \@@_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”.

```

6920     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6921     \@@_create_nodes:
6922     \@@_computations_for_large_nodes:
6923     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6924     \@@_create_nodes:
6925   \endpgfpicture
6926 }

```

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.

```

6927 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6928 {
6929   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6930   \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`.

```

6931   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6932   {
6933     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6934     {
6935       (
6936         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6937         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6938       )
6939       / 2
6940     }
6941     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6942     { l_@@_row _ \@@_i: _ min _ dim }
6943   }
6944   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6945   {
6946     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6947     {
6948       (
6949         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6950         \dim_use:c
6951         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6952       )
6953       / 2
6954     }
6955     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6956     { l_@@_column _ \@@_j: _ max _ dim }
6957   }

```

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

```

6958 \dim_sub:cn
6959 { l_@@_column _ 1 _ min _ dim }
6960 \l_@@_left_margin_dim
6961 \dim_add:cn
6962 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6963 \l_@@_right_margin_dim
6964 }

```

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).

```

6965 \cs_new_protected:Npn \@@_create_nodes:
6966 {
6967   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6968   {
6969     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6970     {

```

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

```

6971       \@@_pgf_rect_node:nnnnn
6972       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6973       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6974       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6975       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6976       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6977       \str_if_empty:NF \l_@@_name_str
6978       {
6979         \pgfnodealias
6980         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6981         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6982       }
6983     }
6984   }

```

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  $n$ .

```

6985   \seq_map_pairwise_function:NNN
6986   \g_@@_multicolumn_cells_seq
6987   \g_@@_multicolumn_sizes_seq
6988   \@@_node_for_multicolumn:nn
6989 }

```

```

6990 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6991 {
6992   \cs_set_nopar:Npn \@@_i: { #1 }
6993   \cs_set_nopar:Npn \@@_j: { #2 }
6994 }

```

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”).

```

6995 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6996 {
6997   \@@_extract_coords_values: #1 \q_stop
6998   \@@_pgf_rect_node:nnnnn
6999   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7000   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

7001 { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7002 { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2-1 } _ max _ dim } }
7003 { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7004 \str_if_empty:NF \l_@@_name_str
7005 {
7006   \pgfnodealias
7007     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7008     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
7009 }
7010 }

```

## 27 The blocks

The following code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass (in the cell of the array).

```

7011 \keys_define:nn { NiceMatrix / Block / FirstPass }
7012 {
7013   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7014   l .value_forbidden:n = true ,
7015   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7016   r .value_forbidden:n = true ,
7017   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7018   c .value_forbidden:n = true ,
7019   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7020   L .value_forbidden:n = true ,
7021   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7022   R .value_forbidden:n = true ,
7023   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7024   C .value_forbidden:n = true ,
7025   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7026   t .value_forbidden:n = true ,
7027   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7028   T .value_forbidden:n = true ,
7029   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7030   b .value_forbidden:n = true ,
7031   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7032   B .value_forbidden:n = true ,
7033   color .code:n =
7034     \@@_color:n { #1 }
7035     \tl_set_rescan:Nnn
7036       \l_@@_draw_tl
7037       { \char_set_catcode_other:N ! }
7038       { #1 } ,
7039   color .value_required:n = true ,
7040   respect-arraystretch .code:n =
7041     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7042   respect-arraystretch .value_forbidden:n = true ,
7043 }

```

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.

```

7044 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7045 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7046 {

```

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).

```

7047 \peek_remove_spaces:n
7048 {
7049   \tl_if_blank:nTF { #2 }
7050   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7051   {
7052     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7053     \@@_Block_i_czech \@@_Block_i
7054     #2 \q_stop
7055   }
7056   { #1 } { #3 } { #4 }
7057 }
7058 }

```

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

```

7059 \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`.

```

7060 {
7061   \char_set_catcode_active:N -
7062   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7063 }

```

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.

```

7064 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7065 {

```

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).

```

7066 \bool_lazy_or:nnTF
7067 { \tl_if_blank_p:n { #1 } }
7068 { \str_if_eq_p:nn { #1 } { * } }
7069 { \int_set:Nn \l_tmpa_int { 100 } }
7070 { \int_set:Nn \l_tmpa_int { #1 } }
7071 \bool_lazy_or:nnTF
7072 { \tl_if_blank_p:n { #2 } }
7073 { \str_if_eq_p:nn { #2 } { * } }
7074 { \int_set:Nn \l_tmpb_int { 100 } }
7075 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7076 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7077 {
7078   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7079   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7080   { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7081 }
7082 { \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.

```

7083 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
7084 \tl_set:Nx \l_tmpa_tl
7085 {
7086   { \int_use:N \c@iRow }
7087   { \int_use:N \c@jCol }
7088   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7089   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7090 }

```

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).

```

7091 \bool_if:nTF
7092 {
7093   (
7094     \int_compare_p:nNn \l_tmpa_int = \c_one_int
7095     ||
7096     \int_compare_p:nNn \l_tmpb_int = \c_one_int
7097   )
7098   && ! \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.

```

7099   && ! \l_@@_X_bool
7100 }
7101 { \exp_args:Nee \@@_Block_iv:nnnnn }
7102 { \exp_args:Nee \@@_Block_v:nnnnn }
7103 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7104 }

```

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.

```

7105 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7106 {
7107   \int_gincr:N \g_@@_block_box_int
7108   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7109   {
7110     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7111     {
7112       \@@_actually_diagbox:nnnnnn
7113       { \int_use:N \c@iRow }
7114       { \int_use:N \c@jCol }
7115       { \int_eval:n { \c@iRow + #1 - 1 } }
7116       { \int_eval:n { \c@jCol + #2 - 1 } }
7117       { \g_@@_row_style_tl \exp_not:n { ##1 } }

```

```

7118         { \g_@@_row_style_tl \exp_not:n { ##2 } }
7119     }
7120 }
7121 \box_gclear_new:c
7122 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7123 \hbox_gset:cn
7124 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7125 {

```

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}`).

```

7126 \tl_if_empty:NTF \l_@@_color_tl
7127 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7128 { \@@_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`.

```

7129 \int_compare:nNnT { #1 } = \c_one_int
7130 {
7131     \int_if_zero:nTF \c@iRow
7132     \l_@@_code_for_first_row_tl
7133     {
7134         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7135         \l_@@_code_for_last_row_tl
7136     }
7137     \g_@@_row_style_tl
7138 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7139 \@@_reset_arraystretch:
7140 \dim_zero:N \extrarowheight

```

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

```

7141 #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.).

```

7142 \@@_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}`.

```

7143 \bool_if:NTF \l_@@_tabular_bool
7144 {
7145     \bool_lazy_all:nTF
7146     {
7147         { \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.

```

7148     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7149     { ! \g_@@_rotate_bool }
7150 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7151     {
7152         \use:e
7153         {
7154             \exp_not:N \begin { minipage }%
7155             [ \str_lowercase:V \l_@@_vpos_block_str ]
7156             { \l_@@_col_width_dim }
7157             \str_case:on \l_@@_hpos_block_str
7158             { c \centering r \raggedleft l \raggedright }
7159         }
7160         #5
7161         \end { minipage }
7162     }

```

In the other cases, we use a `{tabular}`.

```

7163     {
7164         \use:e
7165         {
7166             \exp_not:N \begin { tabular }%
7167             [ \str_lowercase:V \l_@@_vpos_block_str ]
7168             { @ { } \l_@@_hpos_block_str @ { } }
7169         }
7170         #5
7171         \end { tabular }
7172     }
7173 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7174     {
7175         \c_math_toggle_token
7176         \use:e
7177         {
7178             \exp_not:N \begin { array }%
7179             [ \str_lowercase:V \l_@@_vpos_block_str ]
7180             { @ { } \l_@@_hpos_block_str @ { } }
7181         }
7182         #5
7183         \end { array }
7184         \c_math_toggle_token
7185     }
7186 }

```

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).

```

7187     \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.

```

7188     \int_compare:nNnT { #2 } = \c_one_int
7189     {
7190         \dim_gset:Nn \g_@@_blocks_wd_dim
7191         {
7192             \dim_max:nn
7193             \g_@@_blocks_wd_dim
7194             {
7195                 \box_wd:c
7196                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7197             }
7198         }
7199     }

```



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.

```

7200 \str_if_eq:VnT \l_@@_vpos_block_str { c }
7201 {
7202   \int_compare:nNnT { #1 } = \c_one_int
7203   {
7204     \dim_gset:Nn \g_@@_blocks_ht_dim
7205     {
7206       \dim_max:nn
7207       \g_@@_blocks_ht_dim
7208       {
7209         \box_ht:c
7210         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7211       }
7212     }
7213     \dim_gset:Nn \g_@@_blocks_dp_dim
7214     {
7215       \dim_max:nn
7216       \g_@@_blocks_dp_dim
7217       {
7218         \box_dp:c
7219         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7220       }
7221     }
7222   }
7223 }
7224 \seq_gput_right:Nx \g_@@_blocks_seq
7225 {
7226   \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.

```

7227 {
7228   \exp_not:n { #3 } ,
7229   \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7230 \bool_if:NT \g_@@_rotate_bool
7231 {
7232   \bool_if:NTF \g_@@_rotate_c_bool
7233   { m }
7234   { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7235 }
7236
7237 }
7238 {
7239   \box_use_drop:c
7240   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7241 }
7242 }
7243 \bool_set_false:N \g_@@_rotate_c_bool
7244 }

7245 \cs_new:Npn \@@_adjust_hpos_rotate:
7246 {
7247   \bool_if:NT \g_@@_rotate_bool

```

---

<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.

```

7248 {
7249   \str_set:Nx \l_@@_hpos_block_str
7250   {
7251     \bool_if:NTF \g_@@_rotate_c_bool
7252     { c }
7253     {
7254       \str_case:onF \l_@@_vpos_block_str
7255       { b l B l t r T r }
7256       { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7257     }
7258   }
7259 }
7260 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7261 \cs_new_protected:Npn \@@_rotate_box_of_block:
7262 {
7263   \box_grotate:cn
7264   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7265   { 90 }
7266   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7267   {
7268     \vbox_gset_top:cn
7269     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7270     {
7271       \skip_vertical:n { 0.8 ex }
7272       \box_use:c
7273       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7274     }
7275   }
7276   \bool_if:NT \g_@@_rotate_c_bool
7277   {
7278     \hbox_gset:cn
7279     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7280     {
7281       \c_math_toggle_token
7282       \vcenter
7283       {
7284         \box_use:c
7285         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7286       }
7287       \c_math_toggle_token
7288     }
7289   }
7290 }

```

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.

```

7291 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7292 {
7293   \seq_gput_right:Nx \g_@@_blocks_seq
7294   {
7295     \l_tmpa_tl
7296     { \exp_not:n { #3 } }
7297     {
7298       \bool_if:NTF \l_@@_tabular_bool

```

```

7299     {
7300         \group_begin:
The following command will be no-op when respect-arraystretch is in force.
7301         \@@_reset_arraystretch:
7302         \exp_not:n
7303         {
7304             \dim_zero:N \extrarowheight
7305             #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).

```

7306         \bool_if:NT \c_@@_tagging_array_bool { \tag_stop:n { table } }
7307         \use:e
7308         {
7309             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7310             { @ { } \l_@@_hpos_block_str @ { } }
7311         }
7312         #5
7313         \end { tabular }
7314     }
7315     \group_end:
7316 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7317     {
7318         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7319         \@@_reset_arraystretch:
7320         \exp_not:n
7321         {
7322             \dim_zero:N \extrarowheight
7323             #4
7324             \c_math_toggle_token
7325             \use:e
7326             {
7327                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7328                 { @ { } \l_@@_hpos_block_str @ { } }
7329             }
7330             #5
7331             \end { array }
7332             \c_math_toggle_token
7333         }
7334         \group_end:
7335     }
7336 }
7337 }
7338 }

```

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).

```

7339 \keys_define:nn { NiceMatrix / Block / SecondPass }
7340 {
7341     tikz .code:n =
7342         \IfPackageLoadedTF { tikz }
7343         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7344         { \@@_error:n { tikz~key~without~tikz } } ,
7345     tikz .value_required:n = true ,

```

```

7346 fill .code:n =
7347     \tl_set_rescan:Nnn
7348     \l_@@_fill_tl
7349     { \char_set_catcode_other:N ! }
7350     { #1 } ,
7351 fill .value_required:n = true ,
7352 opacity .tl_set:N = \l_@@_opacity_tl ,
7353 opacity .value_required:n = true ,
7354 draw .code:n =
7355     \tl_set_rescan:Nnn
7356     \l_@@_draw_tl
7357     { \char_set_catcode_other:N ! }
7358     { #1 } ,
7359 draw .default:n = default ,
7360 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7361 rounded-corners .default:n = 4 pt ,
7362 color .code:n =
7363     \@@_color:n { #1 }
7364     \tl_set_rescan:Nnn
7365     \l_@@_draw_tl
7366     { \char_set_catcode_other:N ! }
7367     { #1 } ,
7368 borders .clist_set:N = \l_@@_borders_clist ,
7369 borders .value_required:n = true ,
7370 hvlines .meta:n = { vlines , hlines } ,
7371 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7372 vlines .default:n = true ,
7373 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7374 hlines .default:n = true ,
7375 line-width .dim_set:N = \l_@@_line_width_dim ,
7376 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7377 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7378 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7379 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7380 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7381     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7382 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7383     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7384 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7385     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7386 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7387 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7388 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7389 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7390 m .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7391 m .value_forbidden:n = true ,
7392 v-center .meta:n = m ,
7393 name .tl_set:N = \l_@@_block_name_str ,
7394 name .value_required:n = true ,
7395 name .initial:n = ,
7396 respect-arraystretch .code:n =
7397     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7398 respect-arraystretch .value_forbidden:n = true ,
7399 transparent .bool_set:N = \l_@@_transparent_bool ,
7400 transparent .default:n = true ,
7401 transparent .initial:n = false ,
7402 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7403 }

```

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.

```

7404 \cs_new_protected:Npn \@@_draw_blocks:
7405 {
7406   \bool_if:NTF \c_@@_tagging_array_bool
7407     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7408     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7409   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7410 }
7411 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7412 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7413   \int_zero_new:N \l_@@_last_row_int
7414   \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.

```

7415   \int_compare:nNnTF { #3 } > { 99 }
7416     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7417     { \int_set:Nn \l_@@_last_row_int { #3 } }
7418   \int_compare:nNnTF { #4 } > { 99 }
7419     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7420     { \int_set:Nn \l_@@_last_col_int { #4 } }
7421   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7422     {
7423       \bool_lazy_and:nnTF
7424         \l_@@_preamble_bool
7425         {
7426           \int_compare_p:n
7427             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7428         }
7429         {
7430           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7431           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7432           \@@_msg_redirect_name:nn { columns-not-used } { none }
7433         }
7434         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7435     }
7436   {
7437     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7438       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7439       { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7440   }
7441 }

```

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

```

7442 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7443 {

```

The group is for the keys.

```

7444   \group_begin:
7445   \int_compare:nNnT { #1 } = { #3 }
7446     { \str_set:Nn \l_@@_vpos_block_str { t } }
7447   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

```

7448 \bool_if:NT \l_@@_vlines_block_bool
7449 {
7450   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7451   {
7452     \@@_vlines_block:nnn
7453     { \exp_not:n { #5 } }
7454     { #1 - #2 }
7455     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7456   }
7457 }
7458 \bool_if:NT \l_@@_hlines_block_bool
7459 {
7460   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7461   {
7462     \@@_hlines_block:nnn
7463     { \exp_not:n { #5 } }
7464     { #1 - #2 }
7465     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7466   }
7467 }
7468 \bool_if:NF \l_@@_transparent_bool
7469 {
7470   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7471   {

```

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).

```

7472   \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7473   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7474   }
7475 }

```

```

7476 \tl_if_empty:NF \l_@@_draw_tl
7477 {
7478   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7479   { \@@_error:n { hlines~with~color } }
7480   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7481   {
7482     \@@_stroke_block:nnn

```

#5 are the options

```

7483     { \exp_not:n { #5 } }
7484     { #1 - #2 }
7485     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7486   }
7487   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7488   { { #1 } { #2 } { #3 } { #4 } }
7489 }
7490 \clist_if_empty:NF \l_@@_borders_clist
7491 {
7492   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7493   {
7494     \@@_stroke_borders_block:nnn
7495     { \exp_not:n { #5 } }
7496     { #1 - #2 }
7497     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7498   }
7499 }
7500 \tl_if_empty:NF \l_@@_fill_tl
7501 {
7502   \tl_if_empty:NF \l_@@_opacity_tl
7503   {

```

```

7504     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7505     {
7506         \tl_set:Nx \l_@@_fill_tl
7507         {
7508             [ opacity = \l_@@_opacity_tl ,
7509             \tl_tail:o \l_@@_fill_tl
7510         }
7511     }
7512     {
7513         \tl_set:Nx \l_@@_fill_tl
7514         { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7515     }
7516 }
7517 \tl_gput_right:Nx \g_@@_pre_code_before_tl
7518 {
7519     \exp_not:N \roundedrectanglecolor
7520     \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7521         { \l_@@_fill_tl }
7522         { { \l_@@_fill_tl } }
7523         { #1 - #2 }
7524         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7525         { \dim_use:N \l_@@_rounded_corners_dim }
7526     }
7527 }
7528 \seq_if_empty:NF \l_@@_tikz_seq
7529 {
7530     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7531     {
7532         \@@_block_tikz:nnnnn
7533         { #1 }
7534         { #2 }
7535         { \int_use:N \l_@@_last_row_int }
7536         { \int_use:N \l_@@_last_col_int }
7537         { \seq_use:Nn \l_@@_tikz_seq { , } }
7538     }
7539 }
7540 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7541 {
7542     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7543     {
7544         \@@_actually_diagbox:nnnnn
7545         { #1 }
7546         { #2 }
7547         { \int_use:N \l_@@_last_row_int }
7548         { \int_use:N \l_@@_last_col_int }
7549         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7550     }
7551 }
7552 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7553 \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.

```

7554 \pgfpicture
7555 \pgfrememberpicturepositiononpagetrue
7556 \pgf@relevantforpicturesizefalse
7557 \@@_qpoint:n { row - #1 }
7558 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7559 \@@_qpoint:n { col - #2 }
7560 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7561 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7562 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7563 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7564 \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.

```

7565 \@@_pgf_rect_node:nnnnn
7566 { \@@_env: - #1 - #2 - block }
7567 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7568 \str_if_empty:NF \l_@@_block_name_str
7569 {
7570 \pgfnodealias
7571 { \@@_env: - \l_@@_block_name_str }
7572 { \@@_env: - #1 - #2 - block }
7573 \str_if_empty:NF \l_@@_name_str
7574 {
7575 \pgfnodealias
7576 { \l_@@_name_str - \l_@@_block_name_str }
7577 { \@@_env: - #1 - #2 - block }
7578 }
7579 }

```

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.

```

7580 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7581 {
7582 \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.

```

7583 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7584 {

```

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.

```

7585 \cs_if_exist:cT
7586 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7587 {
7588 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7589 {

```



```

7590         \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7591         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7592     }
7593 }
7594 }

```

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.

```

7595     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7596     {
7597         \@@_qpoint:n { col - #2 }
7598         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7599     }
7600     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7601     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7602     {
7603         \cs_if_exist:cT
7604         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7605         {
7606             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7607             {
7608                 \pgfpointanchor
7609                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7610                 { east }
7611                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7612             }
7613         }
7614     }
7615     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7616     {
7617         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7618         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7619     }
7620     \@@_pgf_rect_node:nnnnn
7621     { \@@_env: - #1 - #2 - block - short }
7622     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7623 }

```

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.

```

7624     \bool_if:NT \l_@@_medium_nodes_bool
7625     {
7626         \@@_pgf_rect_node:nnn
7627         { \@@_env: - #1 - #2 - block - medium }
7628         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7629         {
7630             \pgfpointanchor
7631             { \@@_env:
7632               - \int_use:N \l_@@_last_row_int
7633               - \int_use:N \l_@@_last_col_int - medium
7634             }
7635             { south~east }
7636         }
7637     }

```

Now, we will put the label of the block.

```

7638     \bool_lazy_any:nTF
7639     {
7640         { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7641         { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7642         { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7643     }
7644     {

```

If we are in the first column, we must put the block as if it was with the key r.

```
7645 \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.

```
7646 \bool_if:nT \g_@@_last_col_found_bool
7647 {
7648   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7649   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7650 }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
7651 \tl_set:Nx \l_tmpa_tl
7652 {
7653   \str_case:on \l_@@_vpos_block_str
7654   {
7655     c {
7656       \str_case:on \l_@@_hpos_block_str
7657       {
7658         c { center }
7659         l { west }
7660         r { east }
7661       }
7662     }
7663     T {
7664       \str_case:on \l_@@_hpos_block_str
7665       {
7666         c { north }
7667         l { north~west }
7668         r { north~east }
7669       }
7670     }
7671     B {
7672       \str_case:on \l_@@_hpos_block_str
7673       {
7674         c { south }
7675         l { south~west }
7676         r { south~east }
7677       }
7678     }
7679   }
7680 }
7681 }
7682 }
7683 }
7684 \pgftransformshift
7685 {
7686   \pgfpointanchor
7687   {
7688     \@@_env: - #1 - #2 - block
7689     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7690   }
7691   { \l_tmpa_tl }
7692 }
7693 \pgfset
7694 {
7695   inner~xsep = \c_zero_dim ,
7696   inner~ysep = \c_zero_dim
7697 }
7698 \pgfnode
7699 { rectangle }
7700 { \l_tmpa_tl }
7701 { \box_use_drop:N \l_@@_cell_box } { } { }
7702 }
```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

7703 {
7704   \pgfextracty \l_tmpa_dim
7705   {
7706     \@@_qpoint:n
7707     {
7708       row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7709       - base
7710     }
7711   }
7712   \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```

7713   \pgfpointanchor
7714   {
7715     \@@_env: - #1 - #2 - block
7716     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7717   }
7718   {
7719     \str_case:on \l_@@_hpos_block_str
7720     {
7721       c { center }
7722       l { west }
7723       r { east }
7724     }
7725   }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7726   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7727   \pgfset { inner~sep = \c_zero_dim }
7728   \pgfnode
7729   { rectangle }
7730   {
7731     \str_case:on \l_@@_hpos_block_str
7732     {
7733       c { base }
7734       l { base~west }
7735       r { base~east }
7736     }
7737   }
7738   { \box_use_drop:N \l_@@_cell_box } { } { }
7739 }
7740 \endpgfpicture
7741 \group_end:
7742 }

```

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).

```

7743 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7744 {
7745   \group_begin:
7746   \tl_clear:N \l_@@_draw_tl
7747   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7748   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7749   \pgfpicture
7750   \pgfrememberpicturepositiononpagetrue
7751   \pgf@relevantforpicturesizefalse
7752   \tl_if_empty:NF \l_@@_draw_tl
7753   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7754     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7755         { \CT@arc@ }
7756         { \@@_color:o \l_@@_draw_tl }
7757     }
7758     \pgfsetcornersarced
7759     {
7760         \pgfpoint
7761         { \l_@@_rounded_corners_dim }
7762         { \l_@@_rounded_corners_dim }
7763     }
7764     \@@_cut_on_hyphen:w #2 \q_stop
7765     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7766     {
7767         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7768         {
7769             \@@_qpoint:n { row - \l_tmpa_tl }
7770             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7771             \@@_qpoint:n { col - \l_tmpb_tl }
7772             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7773             \@@_cut_on_hyphen:w #3 \q_stop
7774             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7775                 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7776             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7777                 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7778             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7779             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7780             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7781             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7782             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7783             \pgfpathrectanglecorners
7784                 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7785                 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7786             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7787                 { \pgfusepathqstroke }
7788                 { \pgfusepath { stroke } }
7789         }
7790     }
7791     \endpgfpicture
7792     \group_end:
7793 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7794 \keys_define:nn { NiceMatrix / BlockStroke }
7795 {
7796     color .tl_set:N = \l_@@_draw_tl ,
7797     draw .code:n =
7798         \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7799     draw .default:n = default ,
7800     line-width .dim_set:N = \l_@@_line_width_dim ,
7801     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7802     rounded-corners .default:n = 4 pt
7803 }

```

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).

```

7804 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7805 {
7806     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7807     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7808     \@@_cut_on_hyphen:w #2 \q_stop

```

```

7809 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7810 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7811 \@@_cut_on_hyphen:w #3 \q_stop
7812 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7813 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7814 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7815 {
7816   \use:e
7817   {
7818     \@@_vline:n
7819     {
7820       position = ##1 ,
7821       start = \l_@@_tmpc_tl ,
7822       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7823       total-width = \dim_use:N \l_@@_line_width_dim
7824     }
7825   }
7826 }
7827 }
7828 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7829 {
7830   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7831   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7832   \@@_cut_on_hyphen:w #2 \q_stop
7833   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7834   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7835   \@@_cut_on_hyphen:w #3 \q_stop
7836   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7837   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7838   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7839   {
7840     \use:e
7841     {
7842       \@@_hline:n
7843       {
7844         position = ##1 ,
7845         start = \l_@@_tmpd_tl ,
7846         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7847         total-width = \dim_use:N \l_@@_line_width_dim
7848       }
7849     }
7850   }
7851 }

```

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).

```

7852 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7853 {
7854   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7855   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7856   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7857   { \@@_error:n { borders~forbidden } }
7858   {
7859     \tl_clear_new:N \l_@@_borders_tikz_tl
7860     \keys_set:nV
7861     { NiceMatrix / OnlyForTikzInBorders }
7862     \l_@@_borders_clist
7863     \@@_cut_on_hyphen:w #2 \q_stop
7864     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7865     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7866     \@@_cut_on_hyphen:w #3 \q_stop
7867     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }

```

```

7868     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7869     \@@_stroke_borders_block_i:
7870 }
7871 }

7872 \hook_gput_code:nnn { begindocument } { . }
7873 {
7874     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7875     {
7876         \c_@@_pgfortikzpicture_tl
7877         \@@_stroke_borders_block_ii:
7878         \c_@@_endpgfortikzpicture_tl
7879     }
7880 }

7881 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7882 {
7883     \pgfrememberpicturepositiononpagetrue
7884     \pgf@relevantforpicturesizefalse
7885     \CT@arc@
7886     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7887     \clist_if_in:NnT \l_@@_borders_clist { right }
7888     { \@@_stroke_vertical:n \l_tmpb_tl }
7889     \clist_if_in:NnT \l_@@_borders_clist { left }
7890     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7891     \clist_if_in:NnT \l_@@_borders_clist { bottom }
7892     { \@@_stroke_horizontal:n \l_tmpa_tl }
7893     \clist_if_in:NnT \l_@@_borders_clist { top }
7894     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7895 }

7896 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7897 {
7898     tikz .code:n =
7899     \cs_if_exist:NTF \tikzpicture
7900     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7901     { \@@_error:n { tikz~in~borders~without~tikz } } ,
7902     tikz .value_required:n = true ,
7903     top .code:n = ,
7904     bottom .code:n = ,
7905     left .code:n = ,
7906     right .code:n = ,
7907     unknown .code:n = \@@_error:n { bad~border }
7908 }

```

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).

```

7909 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7910 {
7911     \@@_qpoint:n \l_@@_tmpc_tl
7912     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7913     \@@_qpoint:n \l_tmpa_tl
7914     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7915     \@@_qpoint:n { #1 }
7916     \tl_if_empty:NTF \l_@@_borders_tikz_tl
7917     {
7918         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7919         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7920         \pgfusepathqstroke
7921     }
7922     {
7923         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7924         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7925     }
7926 }

```

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).

```

7927 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7928 {
7929   \@@_qpoint:n \l_@@_tmpd_tl
7930   \clist_if_in:NnTF \l_@@_borders_clist { left }
7931     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7932     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7933   \@@_qpoint:n \l_tmpb_tl
7934   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7935   \@@_qpoint:n { #1 }
7936   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7937     {
7938       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7939       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7940       \pgfusepathqstroke
7941     }
7942     {
7943       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7944       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7945     }
7946 }

```

Here is the set of keys for the command \@@\_stroke\_borders\_block:nnn.

```

7947 \keys_define:nn { NiceMatrix / BlockBorders }
7948 {
7949   borders .clist_set:N = \l_@@_borders_clist ,
7950   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7951   rounded-corners .default:n = 4 pt ,
7952   line-width .dim_set:N = \l_@@_line_width_dim
7953 }

```

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.

```

7954 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7955 {
7956   \begin { tikzpicture }
7957   \@@_clip_with_rounded_corners:
7958   \clist_map_inline:nn { #5 }
7959     {
7960       \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7961       \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7962       (
7963         [
7964           xshift = \dim_use:N \l_@@_offset_dim ,
7965           yshift = - \dim_use:N \l_@@_offset_dim
7966         ]
7967         #1 -| #2
7968       )
7969       rectangle
7970       (
7971         [
7972           xshift = - \dim_use:N \l_@@_offset_dim ,
7973           yshift = \dim_use:N \l_@@_offset_dim
7974         ]
7975         \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7976       ) ;
7977     }
7978   \end { tikzpicture }

```

```

7979 }
7980 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7981 \keys_define:nn { NiceMatrix / SpecialOffset }
7982 { offset .dim_set:N = \l_@@_offset_dim }

```

## 28 How to draw the dotted lines transparently

```

7983 \cs_set_protected:Npn \@@_renew_matrix:
7984 {
7985   \RenewDocumentEnvironment { pmatrix } { } {
7986     { \pNiceMatrix }
7987     { \endpNiceMatrix }
7988   }
7989   \RenewDocumentEnvironment { vmatrix } { } {
7990     { \vNiceMatrix }
7991     { \endvNiceMatrix }
7992   }
7993   \RenewDocumentEnvironment { Vmatrix } { } {
7994     { \VNiceMatrix }
7995     { \endVNiceMatrix }
7996   }
7997   \RenewDocumentEnvironment { bmatrix } { } {
7998     { \bNiceMatrix }
7999     { \endbNiceMatrix }
8000   }

```

## 29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8001 \keys_define:nn { NiceMatrix / Auto }
8002 {
8003   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8004   columns-type .value_required:n = true ,
8005   l .meta:n = { columns-type = l } ,
8006   r .meta:n = { columns-type = r } ,
8007   c .meta:n = { columns-type = c } ,
8008   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8009   delimiters / color .value_required:n = true ,
8010   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8011   delimiters / max-width .default:n = true ,
8012   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
8013   delimiters .value_required:n = true ,
8014   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8015   rounded-corners .default:n = 4 pt
8016 }

8017 \NewDocumentCommand \AutoNiceMatrixWithDelims
8018 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8019 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8020 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8021 {

```

The group is for the protection of the keys.

```

8022   \group_begin:
8023   \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8024   \use:e
8025   {

```



```

8026 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8027 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8028 [ \exp_not:o \l_tmpa_tl ]
8029 }
8030 \int_if_zero:nT \l_@@_first_row_int
8031 {
8032 \int_if_zero:nT \l_@@_first_col_int { & }
8033 \prg_replicate:nn { #4 - 1 } { & }
8034 \int_compare:nNtT \l_@@_last_col_int > { -1 } { & } \\
8035 }
8036 \prg_replicate:nn { #3 }
8037 {
8038 \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).

```

8039 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8040 \int_compare:nNtT \l_@@_last_col_int > { -1 } { & } \\
8041 }
8042 \int_compare:nNtT \l_@@_last_row_int > { -2 }
8043 {
8044 \int_if_zero:nT \l_@@_first_col_int { & }
8045 \prg_replicate:nn { #4 - 1 } { & }
8046 \int_compare:nNtT \l_@@_last_col_int > { -1 } { & } \\
8047 }
8048 \end { NiceArrayWithDelims }
8049 \group_end:
8050 }

8051 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8052 {
8053 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8054 {
8055 \bool_gset_true:N \g_@@_delims_bool
8056 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8057 \AutoNiceMatrixWithDelims { #2 } { #3 }
8058 }
8059 }

8060 \@@_define_com:nnn p ( )
8061 \@@_define_com:nnn b [ ]
8062 \@@_define_com:nnn v | |
8063 \@@_define_com:nnn V \ | \ |
8064 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8065 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8066 {
8067 \group_begin:
8068 \bool_gset_false:N \g_@@_delims_bool
8069 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8070 \group_end:
8071 }

```

## 30 The redefinition of the command \dotfill

```

8072 \cs_set_eq:NN \@@_old_dotfill \dotfill
8073 \cs_new_protected:Npn \@@_dotfill:
8074 {

```

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}`).

```
8075 \@@_old_dotfill
8076 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8077 }
```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```
8078 \cs_new_protected:Npn \@@_dotfill_i:
8079 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }
```

## 31 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```
8080 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8081 {
8082   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8083   {
8084     \@@_actually_diagbox:nnnnnn
8085     { \int_use:N \c@iRow }
8086     { \int_use:N \c@jCol }
8087     { \int_use:N \c@iRow }
8088     { \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.

```
8089 { \g_@@_row_style_tl \exp_not:n { #1 } }
8090 { \g_@@_row_style_tl \exp_not:n { #2 } }
8091 }
```

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.

```
8092 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8093 {
8094   { \int_use:N \c@iRow }
8095   { \int_use:N \c@jCol }
8096   { \int_use:N \c@iRow }
8097   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```
8098 { }
8099 }
8100 }
```

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.

```
8101 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8102 {
8103   \pgfpicture
8104   \pgf@relevantforpicturesizefalse
8105   \pgfrememberpicturepositiononpagetrue
8106   \@@_qpoint:n { row - #1 }
8107   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8108   \@@_qpoint:n { col - #2 }
```

```

8109 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8110 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8111 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8112 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8113 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8114 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8115 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8116 {

```

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.

```

8117 \CT@arc@
8118 \pgfsetroundcap
8119 \pgfusepathqstroke
8120 }
8121 \pgfset { inner~sep = 1 pt }
8122 \pgfscope
8123 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8124 \pgfnode { rectangle } { south~west }
8125 {
8126 \begin { minipage } { 20 cm }
8127 \@@_math_toggle: #5 \@@_math_toggle:
8128 \end { minipage }
8129 }
8130 { }
8131 { }
8132 \endpgfscope
8133 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8134 \pgfnode { rectangle } { north~east }
8135 {
8136 \begin { minipage } { 20 cm }
8137 \raggedleft
8138 \@@_math_toggle: #6 \@@_math_toggle:
8139 \end { minipage }
8140 }
8141 { }
8142 { }
8143 \endpgfpicture
8144 }

```

## 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`.

```

8145 \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 `\`.

```

8146 \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`.

```

8147 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8148 {

```

```

8149     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8150     \@@_CodeAfter_iv:n
8151 }

```

We catch the argument of the command `\end` (in `#1`).

```

8152 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8153 {

```

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.

```

8154     \str_if_eq:eeTF \currenvir { #1 }
8155     { \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`.

```

8156     {
8157         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8158         \@@_CodeAfter_ii:n
8159     }
8160 }

```

### 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 columnn. 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.

```

8161 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8162 {
8163     \pgfpicture
8164     \pgfrememberpicturepositiononpagetrue
8165     \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.

```

8166     \@@_qpoint:n { row - 1 }
8167     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8168     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8169     \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).

```

8170     \bool_if:nTF { #3 }
8171     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8172     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8173     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8174     {
8175         \cs_if_exist:cT
8176         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8177         {
8178             \pgfpointanchor
8179             { \@@_env: - ##1 - #2 }
8180             { \bool_if:nTF { #3 } { west } { east } }

```

```

8181         \dim_set:Nn \l_tmpa_dim
8182         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8183     }
8184 }

```

Now we can put the delimiter with a node of PGF.

```

8185     \pgfset { inner~sep = \c_zero_dim }
8186     \dim_zero:N \nulldelimiterspace
8187     \pgftransformshift
8188     {
8189         \pgfpoint
8190         { \l_tmpa_dim }
8191         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8192     }
8193     \pgfnode
8194     { rectangle }
8195     { \bool_if:nTF { #3 } { east } { west } }
8196     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8197         \nullfont
8198         \c_math_toggle_token
8199         \@@_color:o \l_@@_delimiters_color_tl
8200         \bool_if:nTF { #3 } { \left #1 } { \left . }
8201         \vcenter
8202         {
8203             \nullfont
8204             \hrule \@height
8205                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8206                 \@depth \c_zero_dim
8207                 \@width \c_zero_dim
8208         }
8209         \bool_if:nTF { #3 } { \right . } { \right #1 }
8210         \c_math_toggle_token
8211     }
8212     { }
8213     { }
8214 \endpgfpicture
8215 }

```

## 34 The command \SubMatrix

```

8216 \keys_define:nn { NiceMatrix / sub-matrix }
8217 {
8218     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8219     extra-height .value_required:n = true ,
8220     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8221     left-xshift .value_required:n = true ,
8222     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8223     right-xshift .value_required:n = true ,
8224     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8225     xshift .value_required:n = true ,
8226     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8227     delimiters / color .value_required:n = true ,
8228     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8229     slim .default:n = true ,
8230     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8231     hlines .default:n = all ,
8232     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8233     vlines .default:n = all ,
8234     hvlines .meta:n = { hlines, vlines } ,

```

```

8235     hvlines .value_forbidden:n = true
8236   }
8237   \keys_define:nn { NiceMatrix }
8238   {
8239     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8240     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8241     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8242     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8243   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8244   \keys_define:nn { NiceMatrix / SubMatrix }
8245   {
8246     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8247     delimiters / color .value_required:n = true ,
8248     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8249     hlines .default:n = all ,
8250     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8251     vlines .default:n = all ,
8252     hvlines .meta:n = { hlines, vlines } ,
8253     hvlines .value_forbidden:n = true ,
8254     name .code:n =
8255       \tl_if_empty:nTF { #1 }
8256       { \@@_error:n { Invalid-name } }
8257       {
8258         \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8259         {
8260           \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8261           { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8262           {
8263             \str_set:Nn \l_@@_submatrix_name_str { #1 }
8264             \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8265           }
8266         }
8267         { \@@_error:n { Invalid-name } }
8268       } ,
8269     name .value_required:n = true ,
8270     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8271     rules .value_required:n = true ,
8272     code .tl_set:N = \l_@@_code_tl ,
8273     code .value_required:n = true ,
8274     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8275   }

8276   \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8277   {
8278     \peek_remove_spaces:n
8279     {
8280       \tl_gput_right:Nx \g_@@_pre_code_after_tl
8281       {
8282         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8283         [
8284           delimiters / color = \l_@@_delimiters_color_tl ,
8285           hlines = \l_@@_submatrix_hlines_clist ,
8286           vlines = \l_@@_submatrix_vlines_clist ,
8287           extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8288           left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8289           right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8290           slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8291           #5
8292         ]
8293       }

```

```

8294 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8295 }
8296 }
8297 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8298 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8299 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8300 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8301 {
8302   \seq_gput_right:Nx \g_@@_submatrix_seq
8303   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8304 { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8305 { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8306 { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8307 { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8308 }
8309 }

```

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`.

```

8310 \hook_gput_code:nnn { begindocument } { . }
8311 {
8312   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8313   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8314   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8315   {
8316     \peek_remove_spaces:n
8317     {
8318       \@@_sub_matrix:nnnnnnn
8319       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8320     }
8321   }
8322 }

```

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).

```

8323 \NewDocumentCommand \@@_compute_i_j:nn
8324 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8325 { \@@_compute_i_j:nnnn #1 #2 }
8326 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8327 {
8328   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8329   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8330   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8331   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }

```

```

8332 \tl_if_eq:NnT \l_@@_first_i_tl { last }
8333 { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8334 \tl_if_eq:NnT \l_@@_first_j_tl { last }
8335 { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8336 \tl_if_eq:NnT \l_@@_last_i_tl { last }
8337 { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8338 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8339 { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8340 }
8341 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8342 {
8343 \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8344 \@@_compute_i_j:nn { #2 } { #3 }
8345 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8346 { \cs_set_nopar:Npn \arraystretch { 1 } }
8347 \bool_lazy_or:nnTF
8348 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8349 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8350 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8351 {
8352 \str_clear_new:N \l_@@_submatrix_name_str
8353 \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8354 \pgfpicture
8355 \pgfrememberpicturepositiononpagetrue
8356 \pgf@relevantforpicturesizefalse
8357 \pgfset { inner~sep = \c_zero_dim }
8358 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8359 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int\_step\_inline:nnn is provided by curriification.

```

8360 \bool_if:NTF \l_@@_submatrix_slim_bool
8361 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8362 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8363 {
8364 \cs_if_exist:cT
8365 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8366 {
8367 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8368 \dim_set:Nn \l_@@_x_initial_dim
8369 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8370 }
8371 \cs_if_exist:cT
8372 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8373 {
8374 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8375 \dim_set:Nn \l_@@_x_final_dim
8376 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8377 }
8378 }
8379 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8380 { \@@_error:nn { Impossible~delimiter } { left } }
8381 {
8382 \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8383 { \@@_error:nn { Impossible~delimiter } { right } }
8384 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8385 }
8386 \endpgfpicture
8387 }
8388 \group_end:
8389 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.



```

8390 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8391 {
8392   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8393   \dim_set:Nn \l_@@_y_initial_dim
8394   {
8395     \fp_to_dim:n
8396     {
8397       \pgf@y
8398       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8399     }
8400   }
8401   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8402   \dim_set:Nn \l_@@_y_final_dim
8403   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8404   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8405   {
8406     \cs_if_exist:cT
8407     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8408     {
8409       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8410       \dim_set:Nn \l_@@_y_initial_dim
8411       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8412     }
8413     \cs_if_exist:cT
8414     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8415     {
8416       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8417       \dim_set:Nn \l_@@_y_final_dim
8418       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8419     }
8420   }
8421   \dim_set:Nn \l_tmpa_dim
8422   {
8423     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8424     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8425   }
8426   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8427   \group_begin:
8428   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8429   \@@_set_CT@arc@:o \l_@@_rules_color_tl
8430   \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`.

```

8431   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8432   {
8433     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8434     {
8435       \int_compare:nNnT
8436       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8437       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8438         \@@_qpoint:n { col - ##1 }
8439         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8440         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8441         \pgfusepathqstroke
8442       }
8443     }
8444   }

```

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.

```

8445 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8446 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8447 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8448 {
8449   \bool_lazy_and:nnTF
8450   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8451   {
8452     \int_compare_p:nNn
8453     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8454   {
8455     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8456     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8457     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8458     \pgfusepathqstroke
8459   }
8460   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8461 }

```

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.

```

8462 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8463 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8464 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8465 {
8466   \bool_lazy_and:nnTF
8467   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8468   {
8469     \int_compare_p:nNn
8470     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8471   {
8472     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8473 \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

8474 \dim_set:Nn \l_tmpa_dim
8475 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8476 \str_case:nn { #1 }
8477 {
8478   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8479   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8480   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8481   }
8482   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8483 \dim_set:Nn \l_tmpb_dim
8484 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8485 \str_case:nn { #2 }
8486 {
8487   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8488   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8489   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8490 }
8491 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8492 \pgfusepathqstroke
8493 \group_end:
8494 }
8495 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8496 }

```

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).

```

8497   \str_if_empty:NF \l_@@_submatrix_name_str
8498   {
8499     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8500     \l_@@_x_initial_dim \l_@@_y_initial_dim
8501     \l_@@_x_final_dim \l_@@_y_final_dim
8502   }
8503   \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`.

```

8504   \begin { pgfscope }
8505   \pgftransformshift
8506   {
8507     \pgfpoint
8508     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8509     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8510   }
8511   \str_if_empty:NTF \l_@@_submatrix_name_str
8512   { \@@_node_left:nn #1 { } }
8513   { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8514   \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8515   \pgftransformshift
8516   {
8517     \pgfpoint
8518     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8519     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8520   }
8521   \str_if_empty:NTF \l_@@_submatrix_name_str
8522   { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8523   {
8524     \@@_node_right:nnnn #2
8525     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8526   }
8527   \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8528   \flag_clear_new:n { nicematrix }
8529   \l_@@_code_tl
8530 }

```

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`.

```

8531 \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.

```

8532 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8533 {
8534   \use:e
8535   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8536 }

```

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 -.

```
8537 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8538 { #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.

```
8539 \tl_const:Nn \c_@@_integers_alist_tl
8540 {
8541   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8542   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8543   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8544   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8545 }
```

```
8546 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8547 {
```

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`.

```
8548   \tl_if_empty:nTF { #2 }
8549   {
8550     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8551     {
8552       \flag_raise:n { nicematrix }
8553       \int_if_even:nTF { \flag_height:n { nicematrix } }
8554       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8555       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8556     }
8557     { #1 }
8558   }
```

If there is an hyphen, we have to see whether we have a node of the form  $i$ - $j$ , row- $i$  or col- $j$ .

```
8559   { \@@_pgfpointanchor_iii:w { #1 } #2 }
8560 }
```

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`).

```
8561 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8562 {
8563   \str_case:nnF { #1 }
8564   {
8565     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8566     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8567   }
```

Now the case of a node of the form  $i$ - $j$ .

```
8568   {
8569     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8570     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8571   }
8572 }
```

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`).

```

8573 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8574 {
8575   \pgfnode
8576   { rectangle }
8577   { east }
8578   {
8579     \nullfont
8580     \c_math_toggle_token
8581     \@@_color:o \l_@@_delimiters_color_tl
8582     \left #1
8583     \vcenter
8584     {
8585       \nullfont
8586       \hrule \@height \l_tmpa_dim
8587               \@depth \c_zero_dim
8588               \@width \c_zero_dim
8589     }
8590     \right .
8591     \c_math_toggle_token
8592   }
8593   { #2 }
8594   { }
8595 }

```

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.

```

8596 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8597 {
8598   \pgfnode
8599   { rectangle }
8600   { west }
8601   {
8602     \nullfont
8603     \c_math_toggle_token
8604     \@@_color:o \l_@@_delimiters_color_tl
8605     \left .
8606     \vcenter
8607     {
8608       \nullfont
8609       \hrule \@height \l_tmpa_dim
8610               \@depth \c_zero_dim
8611               \@width \c_zero_dim
8612     }
8613     \right #1
8614     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8615     ^ { \smash { #4 } }
8616     \c_math_toggle_token
8617   }
8618   { #2 }
8619   { }
8620 }

```

## 35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8621 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8622 {
8623   \peek_remove_spaces:n
8624   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8625 }
8626 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8627 {
8628   \peek_remove_spaces:n
8629   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8630 }
8631 \keys_define:nn { NiceMatrix / Brace }
8632 {
8633   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8634   left-shorten .default:n = true ,
8635   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8636   shorten .meta:n = { left-shorten , right-shorten } ,
8637   right-shorten .default:n = true ,
8638   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8639   yshift .value_required:n = true ,
8640   yshift .initial:n = \c_zero_dim ,
8641   color .tl_set:N = \l_tmpa_tl ,
8642   color .value_required:n = true ,
8643   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8644 }

```

#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*.

```

8645 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8646 {
8647   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8648 \@@_compute_i_j:nn { #1 } { #2 }
8649 \bool_lazy_or:nnTF
8650 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8651 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8652 {
8653   \str_if_eq:nnTF { #5 } { under }
8654   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8655   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8656 }
8657 {
8658   \tl_clear:N \l_tmpa_tl
8659   \keys_set:nn { NiceMatrix / Brace } { #4 }
8660   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8661   \pgfpicture
8662   \pgfrememberpicturepositiononpagetrue
8663   \pgf@relevantforpicturesizefalse
8664   \bool_if:NT \l_@@_brace_left_shorten_bool
8665   {
8666     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8667     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8668     {
8669       \cs_if_exist:cT
8670       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8671       {
8672         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8673         \dim_set:Nn \l_@@_x_initial_dim
8674         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8675       }
8676     }

```

```

8677     }
8678     \bool_lazy_or:nnT
8679     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8680     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8681     {
8682         \@@_qpoint:n { col - \l_@@_first_j_tl }
8683         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8684     }
8685     \bool_if:NT \l_@@_brace_right_shorten_bool
8686     {
8687         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8688         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8689         {
8690             \cs_if_exist:cT
8691             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8692             {
8693                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8694                 \dim_set:Nn \l_@@_x_final_dim
8695                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8696             }
8697         }
8698     }
8699     \bool_lazy_or:nnT
8700     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8701     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8702     {
8703         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8704         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8705     }
8706     \pgfset { inner~sep = \c_zero_dim }
8707     \str_if_eq:nnTF { #5 } { under }
8708     { \@@_underbrace_i:n { #3 } }
8709     { \@@_overbrace_i:n { #3 } }
8710     \endpgfpicture
8711 }
8712 \group_end:
8713 }

```

The argument is the text to put above the brace.

```

8714 \cs_new_protected:Npn \@@_overbrace_i:n #1
8715 {
8716     \@@_qpoint:n { row - \l_@@_first_i_tl }
8717     \pgftransformshift
8718     {
8719         \pgfpoint
8720         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8721         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8722     }
8723     \pgfnode
8724     { rectangle }
8725     { south }
8726     {
8727         \vtop
8728         {
8729             \group_begin:
8730             \everycr { }
8731             \halign
8732             {
8733                 \hfil ## \hfil \crrc
8734                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8735                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8736                 \c_math_toggle_token
8737                 \overbrace
8738                 {

```

```

8739             \hbox_to_wd:nn
8740             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8741             { }
8742         }
8743         \c_math_toggle_token
8744         \cr
8745     }
8746     \group_end:
8747 }
8748 }
8749 { }
8750 { }
8751 }

```

The argument is the text to put under the brace.

```

8752 \cs_new_protected:Npn \@@_underbrace_i:n #1
8753 {
8754     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8755     \pgftransformshift
8756     {
8757         \pgfpoint
8758         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8759         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8760     }
8761     \pgfnode
8762     { rectangle }
8763     { north }
8764     {
8765         \group_begin:
8766         \everycr { }
8767         \vbox
8768         {
8769             \halign
8770             {
8771                 \hfil ## \hfil \crrc
8772                 \c_math_toggle_token
8773                 \underbrace
8774                 {
8775                     \hbox_to_wd:nn
8776                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8777                     { }
8778                 }
8779                 \c_math_toggle_token
8780                 \cr
8781                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8782                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8783             }
8784         }
8785         \group_end:
8786     }
8787     { }
8788     { }
8789 }

```

## 36 The command TikzEveryCell

```

8790 \bool_new:N \l_@@_not_empty_bool
8791 \bool_new:N \l_@@_empty_bool
8792

```



```

8793 \keys_define:nn { NiceMatrix / TikzEveryCell }
8794 {
8795     not-empty .code:n =
8796         \bool_lazy_or:nnTF
8797             \l_@@_in_code_after_bool
8798             \g_@@_recreate_cell_nodes_bool
8799             { \bool_set_true:N \l_@@_not_empty_bool }
8800             { \@@_error:n { detection-of-empty-cells } } } ,
8801     not-empty .value_forbidden:n = true ,
8802     empty .code:n =
8803         \bool_lazy_or:nnTF
8804             \l_@@_in_code_after_bool
8805             \g_@@_recreate_cell_nodes_bool
8806             { \bool_set_true:N \l_@@_empty_bool }
8807             { \@@_error:n { detection-of-empty-cells } } } ,
8808     empty .value_forbidden:n = true ,
8809     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
8810 }
8811
8812
8813 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8814 {
8815     \IfPackageLoadedTF { tikz }
8816     {
8817         \group_begin:
8818         \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.

```

8819         \tl_set:Nn \l_tmpa_tl { { #2 } }
8820         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8821             { \@@_for_a_block:nnnnn ##1 }
8822         \@@_all_the_cells:
8823         \group_end:
8824     }
8825     { \@@_error:n { TikzEveryCell-without-tikz } }
8826 }
8827
8828 \tl_new:N \@@_i_tl
8829 \tl_new:N \@@_j_tl
8830
8831 \cs_new_protected:Nn \@@_all_the_cells:
8832 {
8833     \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8834     {
8835         \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8836         {
8837             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8838             {
8839                 \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8840                     { \@@_i_tl - \@@_j_tl }
8841                 {
8842                     \bool_set_false:N \l_tmpa_bool
8843                     \cs_if_exist:cTF
8844                         { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8845                         {
8846                             \bool_if:NF \l_@@_empty_bool
8847                             { \bool_set_true:N \l_tmpa_bool }
8848                         }
8849                     {
8850                         \bool_if:NF \l_@@_not_empty_bool
8851                         { \bool_set_true:N \l_tmpa_bool }
8852                     }
8853                     \bool_if:NT \l_tmpa_bool

```

```

8854         {
8855             \@@_block_tikz:nnnnV
8856             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8857         }
8858     }
8859 }
8860 }
8861 }
8862 }
8863
8864 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8865 {
8866     \bool_if:NF \l_@@_empty_bool
8867     {
8868         \@@_block_tikz:nnnnV
8869         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8870     }
8871     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8872 }
8873
8874 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8875 {
8876     \int_step_inline:nnn { #1 } { #3 }
8877     {
8878         \int_step_inline:nnn { #2 } { #4 }
8879         { \cs_set:cpn { cell - ##1 - #####1 } { } }
8880     }
8881 }

```

## 37 The command \ShowCellNames

```

8882 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8883 {
8884     \dim_gzero_new:N \g_@@_tmpc_dim
8885     \dim_gzero_new:N \g_@@_tmpd_dim
8886     \dim_gzero_new:N \g_@@_tmpe_dim
8887     \int_step_inline:nn \c@iRow
8888     {
8889         \begin { pgfpicture }
8890         \@@_qpoint:n { row - ##1 }
8891         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8892         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8893         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8894         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8895         \bool_if:NTF \l_@@_in_code_after_bool
8896         \end { pgfpicture }
8897         \int_step_inline:nn \c@jCol
8898         {
8899             \hbox_set:Nn \l_tmpa_box
8900             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8901             \begin { pgfpicture }
8902             \@@_qpoint:n { col - #####1 }
8903             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8904             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8905             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8906             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8907             \endpgfpicture
8908             \end { pgfpicture }
8909             \fp_set:Nn \l_tmpa_fp
8910             {
8911                 \fp_min:nn
8912                 {

```

```

8913         \fp_min:nn
8914         {
8915             \dim_ratio:nn
8916             { \g_@@_tmpd_dim }
8917             { \box_wd:N \l_tmpa_box }
8918         }
8919         {
8920             \dim_ratio:nn
8921             { \g_tmpb_dim }
8922             { \box_ht_plus_dp:N \l_tmpa_box }
8923         }
8924     }
8925     { 1.0 }
8926 }
8927 \box_scale:Nnn \l_tmpa_box
8928 { \fp_use:N \l_tmpa_fp }
8929 { \fp_use:N \l_tmpa_fp }
8930 \pgfpicture
8931 \pgfrememberpicturepositiononpagetrue
8932 \pgf@relevantforpicturesizefalse
8933 \pgftransformshift
8934 {
8935     \pgfpoint
8936     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8937     { \dim_use:N \g_tmpa_dim }
8938 }
8939 \pgfnode
8940 { rectangle }
8941 { center }
8942 { \box_use:N \l_tmpa_box }
8943 { }
8944 { }
8945 \endpgfpicture
8946 }
8947 }
8948 }
8949 \NewDocumentCommand \@@_ShowCellNames { }
8950 {
8951     \bool_if:NT \l_@@_in_code_after_bool
8952     {
8953         \pgfpicture
8954         \pgfrememberpicturepositiononpagetrue
8955         \pgf@relevantforpicturesizefalse
8956         \pgfpathrectanglecorners
8957         { \@@_qpoint:n { 1 } }
8958         {
8959             \@@_qpoint:n
8960             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8961         }
8962         \pgfsetfillopacity { 0.75 }
8963         \pgfsetfillcolor { white }
8964         \pgfusepathqfill
8965         \endpgfpicture
8966     }
8967     \dim_gzero_new:N \g_@@_tmpc_dim
8968     \dim_gzero_new:N \g_@@_tmpd_dim
8969     \dim_gzero_new:N \g_@@_tmpe_dim
8970     \int_step_inline:nn \c@iRow
8971     {
8972         \bool_if:NTF \l_@@_in_code_after_bool
8973         {
8974             \pgfpicture
8975             \pgfrememberpicturepositiononpagetrue

```

```

8976         \pgf@relevantforpicturesizefalse
8977     }
8978     { \begin { pgfpicture } }
8979     \@@_qpoint:n { row - ##1 }
8980     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8981     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8982     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8983     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8984     \bool_if:NTF \l_@@_in_code_after_bool
8985     { \endpgfpicture }
8986     { \end { pgfpicture } }
8987     \int_step_inline:nn \c@jCol
8988     {
8989         \hbox_set:Nn \l_tmpa_box
8990         {
8991             \normalfont \Large \sffamily \bfseries
8992             \bool_if:NTF \l_@@_in_code_after_bool
8993             { \color { red } }
8994             { \color { red ! 50 } }
8995             ##1 - ####1
8996         }
8997         \bool_if:NTF \l_@@_in_code_after_bool
8998         {
8999             \pgfpicture
9000             \pgfrememberpicturerepositiononpagetrue
9001             \pgf@relevantforpicturesizefalse
9002         }
9003         { \begin { pgfpicture } }
9004         \@@_qpoint:n { col - ####1 }
9005         \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9006         \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9007         \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9008         \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9009         \bool_if:NTF \l_@@_in_code_after_bool
9010         { \endpgfpicture }
9011         { \end { pgfpicture } }
9012         \fp_set:Nn \l_tmpa_fp
9013         {
9014             \fp_min:nn
9015             {
9016                 \fp_min:nn
9017                 { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9018                 { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9019             }
9020             { 1.0 }
9021         }
9022         \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9023         \pgfpicture
9024         \pgfrememberpicturerepositiononpagetrue
9025         \pgf@relevantforpicturesizefalse
9026         \pgftransformshift
9027         {
9028             \pgfpoint
9029             { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9030             { \dim_use:N \g_tmpa_dim }
9031         }
9032         \pgfnode
9033         { rectangle }
9034         { center }
9035         { \box_use:N \l_tmpa_box }
9036         { }
9037         { }
9038         \endpgfpicture

```

```

9039     }
9040   }
9041 }

```

## 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.

```

9042 \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.

```

9043 \bool_new:N \g_@@_footnote_bool

9044 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9045 {
9046   The~key~'\l_keys_key_str'~is~unknown. \\
9047   That~key~will~be~ignored. \\
9048   For~a~list~of~the~available~keys,~type~H~<return>.
9049 }
9050 {
9051   The~available~keys~are~(in~alphabetic~order):~
9052   footnote,~
9053   footnotehyper,~
9054   messages~for~Overleaf,~
9055   no~test~for~array,~
9056   renew~dots,~and~
9057   renew~matrix.
9058 }

9059 \keys_define:nn { NiceMatrix / Package }
9060 {
9061   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9062   renew~dots .value_forbidden:n = true ,
9063   renew~matrix .code:n = \@@_renew_matrix: ,
9064   renew~matrix .value_forbidden:n = true ,
9065   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9066   footnote .bool_set:N = \g_@@_footnote_bool ,
9067   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9068   no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
9069   no~test~for~array .default:n = true ,
9070   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9071 }
9072 \ProcessKeysOptions { NiceMatrix / Package }

9073 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9074 {
9075   You~can't~use~the~option~'footnote'~because~the~package~
9076   footnotehyper~has~already~been~loaded.~
9077   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9078   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9079   of~the~package~footnotehyper.\\
9080   The~package~footnote~won't~be~loaded.
9081 }

```

```

9082 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9083 {
9084   You~can't~use~the~option~'footnotehyper'~because~the~package~
9085   footnote~has~already~been~loaded.~
9086   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9087   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9088   of~the~package~footnote.\\
9089   The~package~footnotehyper~won't~be~loaded.
9090 }

9091 \bool_if:NT \g_@@_footnote_bool
9092 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9093   \IfClassLoadedTF { beamer }
9094   { \bool_set_false:N \g_@@_footnote_bool }
9095   {
9096     \IfPackageLoadedTF { footnotehyper }
9097     { \@@_error:n { footnote~with~footnotehyper~package } }
9098     { \usepackage { footnote } }
9099   }
9100 }

9101 \bool_if:NT \g_@@_footnotehyper_bool
9102 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9103   \IfClassLoadedTF { beamer }
9104   { \bool_set_false:N \g_@@_footnote_bool }
9105   {
9106     \IfPackageLoadedTF { footnote }
9107     { \@@_error:n { footnotehyper~with~footnote~package } }
9108     { \usepackage { footnotehyper } }
9109   }
9110   \bool_set_true:N \g_@@_footnote_bool
9111 }

```

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.

```

9112 \bool_new:N \l_@@_underscore_loaded_bool
9113 \IfPackageLoadedTF { underscore }
9114 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9115 { }

9116 \hook_gput_code:nnn { begindocument } { . }
9117 {
9118   \bool_if:NF \l_@@_underscore_loaded_bool
9119   {
9120     \IfPackageLoadedTF { underscore }
9121     { \@@_error:n { underscore~after~nicematrix } }
9122     { }
9123   }
9124 }

```

## 40 Error messages of the package

```

9125 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9126 { \str_const:Nn \c_@@_available_keys_str { } }
9127 {
9128   \str_const:Nn \c_@@_available_keys_str
9129   { For~a~list~of~the~available~keys,~type~H~<return>. }
9130 }

9131 \seq_new:N \g_@@_types_of_matrix_seq
9132 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9133 {
9134   NiceMatrix ,
9135   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9136 }
9137 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9138 { \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`.

```

9139 \cs_new_protected:Npn \@@_error_too_much_cols:
9140 {
9141   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9142   {
9143     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9144     { \@@_fatal:n { too~much~cols~for~matrix } }
9145     {
9146       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9147       { \@@_fatal:n { too~much~cols~for~matrix } }
9148       {
9149         \bool_if:NF \l_@@_last_col_without_value_bool
9150         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9151       }
9152     }
9153   }
9154   { \@@_fatal:nn { too~much~cols~for~array } }
9155 }

```

The following command must *not* be protected since it's used in an error message.

```

9156 \cs_new:Npn \@@_message_hdotsfor:
9157 {
9158   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9159   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9160 }

9161 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9162 {
9163   Incompatible~options.\\
9164   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9165   The~output~will~not~be~reliable.
9166 }

9167 \@@_msg_new:nn { negative~weight }
9168 {
9169   Negative~weight.\\
9170   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9171   the~value~'\int_use:N \l_@@_weight_int'.\\
9172   The~absolute~value~will~be~used.
9173 }

9174 \@@_msg_new:nn { last~col~not~used }
9175 {
9176   Column~not~used.\\

```

```

9177 The-key~'last-col'~is-in-force-but~you-have-not-used-that-last-column~
9178 in-your~\@@_full_name_env:~.~However,~you~can~go-on.
9179 }

9180 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9181 {
9182   Too-much-columns.\\
9183   In-the-row~\int_eval:n { \c@iRow },~
9184   you-try-to-use-more-columns~
9185   than-allowed-by-your~\@@_full_name_env:~\@@_message_hdotsfor:\
9186   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9187   (plus-the-exterior-columns).~This-error-is-fatal.
9188 }

9189 \@@_msg_new:nn { too-much-cols-for-matrix }
9190 {
9191   Too-much-columns.\\
9192   In-the-row~\int_eval:n { \c@iRow },~
9193   you-try-to-use-more-columns-than-allowed-by-your~
9194   \@@_full_name_env:~\@@_message_hdotsfor:\ Recall~that~the~maximal~
9195   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9196   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9197   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9198   \token_to_str:N \setcounter\ to~change~that~value).~
9199   This-error-is-fatal.
9200 }

9201 \@@_msg_new:nn { too-much-cols-for-array }
9202 {
9203   Too-much-columns.\\
9204   In-the-row~\int_eval:n { \c@iRow },~
9205   ~you-try-to-use-more-columns-than-allowed-by-your~
9206   \@@_full_name_env:~\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9207   \int_use:N \g_@@_static_num_of_col_int\
9208   ~(plus~the~potential~exterior~ones).
9209   This-error-is-fatal.
9210 }

9211 \@@_msg_new:nn { columns-not-used }
9212 {
9213   Columns-not-used.\\
9214   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9215   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9216   The-columns-you-did-not-used-won't-be-created.\\
9217   You-won't-have-similar-error-message-till-the-end-of-the-document.
9218 }

9219 \@@_msg_new:nn { in-first-col }
9220 {
9221   Erroneous-use.\\
9222   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9223   That-command-will-be-ignored.
9224 }

9225 \@@_msg_new:nn { in-last-col }
9226 {
9227   Erroneous-use.\\
9228   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9229   That-command-will-be-ignored.
9230 }

9231 \@@_msg_new:nn { in-first-row }
9232 {
9233   Erroneous-use.\\
9234   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9235   That-command-will-be-ignored.
9236 }

```



```

9237 \@@_msg_new:nn { in~last~row }
9238 {
9239   You~can't~use~the~command~\#1 in~the~last~row~(exterior)~of~the~array.\\
9240   That~command~will~be~ignored.
9241 }
9242 \@@_msg_new:nn { caption~outside~float }
9243 {
9244   Key~caption~forbidden.\\
9245   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9246   environment.~This~key~will~be~ignored.
9247 }
9248 \@@_msg_new:nn { short~caption~without~caption }
9249 {
9250   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9251   However,~your~'short~caption'~will~be~used~as~'caption'.
9252 }
9253 \@@_msg_new:nn { double~closing~delimiter }
9254 {
9255   Double~delimiter.\\
9256   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9257   delimiter.~This~delimiter~will~be~ignored.
9258 }
9259 \@@_msg_new:nn { delimiter~after~opening }
9260 {
9261   Double~delimiter.\\
9262   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9263   delimiter.~That~delimiter~will~be~ignored.
9264 }
9265 \@@_msg_new:nn { bad~option~for~line~style }
9266 {
9267   Bad~line~style.\\
9268   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9269   is~'standard'.~That~key~will~be~ignored.
9270 }
9271 \@@_msg_new:nn { Identical~notes~in~caption }
9272 {
9273   Identical~tabular~notes.\\
9274   You~can't~put~several~notes~with~the~same~content~in~
9275   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9276   If~you~go~on,~the~output~will~probably~be~erroneous.
9277 }
9278 \@@_msg_new:nn { tabularnote~below~the~tabular }
9279 {
9280   \token_to_str:N \tabularnote\ forbidden\\
9281   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9282   of~your~tabular~because~the~caption~will~be~composed~below~
9283   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9284   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9285   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9286   no~similar~error~will~raised~in~this~document.
9287 }
9288 \@@_msg_new:nn { Unknown~key~for~rules }
9289 {
9290   Unknown~key.\\
9291   There~is~only~two~keys~available~here:~width~and~color.\\
9292   Your~key~'\l_keys_key_str'~will~be~ignored.
9293 }
9294 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9295 {
9296   Unknown~key.\\

```

```

9297     There~is~only~two~keys~available~here:~
9298     'empty'~and~'not-empty'.\\
9299     Your~key~'\l_keys_key_str'~will~be~ignored.
9300 }

9301 \@@_msg_new:nn { Unknown~key~for~rotate }
9302 {
9303     Unknown~key.\\
9304     The~only~key~available~here~is~'c'.\\
9305     Your~key~'\l_keys_key_str'~will~be~ignored.
9306 }

9307 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9308 {
9309     Unknown~key.\\
9310     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9311     It~you~go~on,~you~will~probably~have~other~errors. \\
9312     \c_@@_available_keys_str
9313 }
9314 {
9315     The~available~keys~are~(in~alphabetic~order):~
9316     ccommand,~
9317     color,~
9318     command,~
9319     dotted,~
9320     letter,~
9321     multiplicity,~
9322     sep-color,~
9323     tikz,~and~total-width.
9324 }

9325 \@@_msg_new:nnn { Unknown~key~for~xdots }
9326 {
9327     Unknown~key.\\
9328     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9329     \c_@@_available_keys_str
9330 }
9331 {
9332     The~available~keys~are~(in~alphabetic~order):~
9333     'color',~
9334     'horizontal-labels',~
9335     'inter',~
9336     'line-style',~
9337     'radius',~
9338     'shorten',~
9339     'shorten-end'~and~'shorten-start'.
9340 }

9341 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9342 {
9343     Unknown~key.\\
9344     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9345     (and~you~try~to~use~'\l_keys_key_str')\\
9346     That~key~will~be~ignored.
9347 }

9348 \@@_msg_new:nn { label~without~caption }
9349 {
9350     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9351     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9352 }

9353 \@@_msg_new:nn { W~warning }
9354 {
9355     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9356     (row~\int_use:N \c@iRow).
9357 }

```

```

9358 \@@_msg_new:nn { Construct-too-large }
9359 {
9360   Construct-too-large.\
9361   Your~command~\token_to_str:N #1
9362   can't~be~drawn~because~your~matrix~is~too~small.\
9363   That~command~will~be~ignored.
9364 }
9365 \@@_msg_new:nn { underscore-after-nicematrix }
9366 {
9367   Problem-with~'underscore'.\
9368   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9369   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\
9370   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9371 }
9372 \@@_msg_new:nn { ampersand-in-light-syntax }
9373 {
9374   Ampersand~forbidden.\
9375   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9376   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9377 }
9378 \@@_msg_new:nn { double-backslash-in-light-syntax }
9379 {
9380   Double~backslash~forbidden.\
9381   You~can't~use~\token_to_str:N
9382   \~to~separate~rows~because~the~key~'light-syntax'~
9383   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9384   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9385 }
9386 \@@_msg_new:nn { hlines-with-color }
9387 {
9388   Incompatible~keys.\
9389   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9390   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\
9391   Maybe~it~will~possible~in~future~version.\
9392   Your~key~will~be~discarded.
9393 }
9394 \@@_msg_new:nn { bad-value-for-baseline }
9395 {
9396   Bad~value~for~baseline.\
9397   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9398   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9399   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9400   the~form~'line-i'.\
9401   A~value~of~1~will~be~used.
9402 }
9403 \@@_msg_new:nn { detection-of-empty-cells }
9404 {
9405   Problem-with~'not-empty'\
9406   For~technical~reasons,~you~must~activate~
9407   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9408   in~order~to~use~the~key~'\l_keys_key_str'.\
9409   That~key~will~be~ignored.
9410 }
9411 \@@_msg_new:nn { siunitx-not-loaded }
9412 {
9413   siunitx~not~loaded\
9414   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\
9415   That~error~is~fatal.
9416 }
9417 \@@_msg_new:nn { ragged2e-not-loaded }

```

```

9418 {
9419     You-have-to-load~'ragged2e'~in-order-to-use-the-key~'\l_keys_key_str'~in~
9420     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The-key~'\str_lowercase:V
9421     \l_keys_key_str'~will-be-used-instead.
9422 }
9423 \@@_msg_new:nn { Invalid-name }
9424 {
9425     Invalid-name.\\
9426     You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
9427     \SubMatrix\ of~your~\@@_full_name_env:~.\\
9428     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9429     This-key-will-be-ignored.
9430 }
9431 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9432 {
9433     Wrong-line.\\
9434     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
9435     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but-that~
9436     number-is-not-valid.~It-will-be-ignored.
9437 }
9438 \@@_msg_new:nn { Impossible-delimiter }
9439 {
9440     Impossible-delimiter.\\
9441     It's-impossible-to-draw-the~#1~delimiter-of~your~
9442     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9443     in-that-column.
9444     \bool_if:NT \l_@@_submatrix_slim_bool
9445     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9446     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9447 }
9448 \@@_msg_new:nnn { width-without-X-columns }
9449 {
9450     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9451     That-key-will-be-ignored.
9452 }
9453 {
9454     This-message-is-the-message~'width-without-X-columns'~
9455     of-the-module~'nicematrix'.~
9456     The-experimented-users-can-disable-that-message-with~
9457     \token_to_str:N \msg_redirect_name:nnn.\\
9458 }
9459
9460 \@@_msg_new:nn { key-multiplicity-with-dotted }
9461 {
9462     Incompatible-keys. \\
9463     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9464     in-a~'custom-line'.~They-are-incompatible. \\
9465     The-key~'multiplicity'~will-be-discarded.
9466 }
9467 \@@_msg_new:nn { empty-environment }
9468 {
9469     Empty-environment.\\
9470     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9471 }
9472 \@@_msg_new:nn { No-letter-and-no-command }
9473 {
9474     Erroneous-use.\\
9475     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9476     key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9477     ~'ccommand'~(to-draw-horizontal-rules).\\
9478     However,~you-can-go-on.

```

```

9479 }
9480 \@@_msg_new:nn { Forbidden-letter }
9481 {
9482   Forbidden-letter.\
9483   You-can't-use-the-letter~'#1'~for-a-customized-line.\
9484   It-will-be-ignored.
9485 }
9486 \@@_msg_new:nn { Several-letters }
9487 {
9488   Wrong-name.\
9489   You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you-
9490   have-used~'\l_@@_letter_str').\
9491   It-will-be-ignored.
9492 }
9493 \@@_msg_new:nn { Delimiter-with-small }
9494 {
9495   Delimiter~forbidden.\
9496   You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\
9497   because-the-key~'small'~is-in-force.\
9498   This-error-is-fatal.
9499 }
9500 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9501 {
9502   Unknown-cell.\
9503   Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-
9504   the~\token_to_str:N \CodeAfter\ of-your~\@@_full_name_env:\
9505   can't-be-executed-because-a-cell-doesn't-exist.\
9506   This-command~\token_to_str:N \line\ will-be-ignored.
9507 }
9508 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9509 {
9510   Duplicate-name.\
9511   The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
9512   in-this~\@@_full_name_env:.\
9513   This-key-will-be-ignored.\
9514   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9515   { For-a-list-of-the-names-already-used,~type-H<return>. }
9516 }
9517 {
9518   The-names-already-defined-in-this~\@@_full_name_env:\ are:~
9519   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9520 }
9521 \@@_msg_new:nn { r-or-l-with-preamble }
9522 {
9523   Erroneous-use.\
9524   You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~
9525   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
9526   your~\@@_full_name_env:.\
9527   This-key-will-be-ignored.
9528 }
9529 \@@_msg_new:nn { Hdotsfor-in-col-0 }
9530 {
9531   Erroneous-use.\
9532   You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior-column-of-
9533   the-array.~This-error-is-fatal.
9534 }
9535 \@@_msg_new:nn { bad-corner }
9536 {
9537   Bad-corner.\
9538   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~

```

```

9539     'corners')~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9540     This~specification~of~corner~will~be~ignored.
9541 }
9542 \@@_msg_new:nn { bad~border }
9543 {
9544     Bad~border.\\
9545     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9546     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9547     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9548     also~use~the~key~'tikz'
9549     \IfPackageLoadedTF { tikz }
9550         { }
9551         {~if~you~load~the~LaTeX~package~'tikz'}).\\
9552     This~specification~of~border~will~be~ignored.
9553 }
9554 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9555 {
9556     TikZ~not~loaded.\\
9557     You~can't~use~\token_to_str:N \TikzEveryCell\
9558     because~you~have~not~loaded~tikz.~
9559     This~command~will~be~ignored.
9560 }
9561 \@@_msg_new:nn { tikz~key~without~tikz }
9562 {
9563     TikZ~not~loaded.\\
9564     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9565     \Block'~because~you~have~not~loaded~tikz.~
9566     This~key~will~be~ignored.
9567 }
9568 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9569 {
9570     Erroneous~use.\\
9571     In~the~\@@_full_name_env:,~you~must~use~the~key~
9572     'last~col'~without~value.\\
9573     However,~you~can~go~on~for~this~time~
9574     (the~value~'\l_keys_value_tl'~will~be~ignored).
9575 }
9576 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9577 {
9578     Erroneous~use.\\
9579     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9580     'last~col'~without~value.\\
9581     However,~you~can~go~on~for~this~time~
9582     (the~value~'\l_keys_value_tl'~will~be~ignored).
9583 }
9584 \@@_msg_new:nn { Block~too~large~1 }
9585 {
9586     Block~too~large.\\
9587     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9588     too~small~for~that~block. \\
9589     This~block~and~maybe~others~will~be~ignored.
9590 }
9591 \@@_msg_new:nn { Block~too~large~2 }
9592 {
9593     Block~too~large.\\
9594     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9595     \g_@@_static_num_of_col_int\
9596     columns~but~you~use~only~\int_use:N \c_jCol\ and~that's~why~a~block~
9597     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9598     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:~\\
9599     This~block~and~maybe~others~will~be~ignored.

```

```

9600 }
9601 \@@_msg_new:nn { unknown~column~type }
9602 {
9603   Bad~column~type.\\
9604   The~column~type~'#1'~in~your~\@@_full_name_env:\
9605   is~unknown. \\
9606   This~error~is~fatal.
9607 }
9608 \@@_msg_new:nn { unknown~column~type~S }
9609 {
9610   Bad~column~type.\\
9611   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9612   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9613   load~that~package. \\
9614   This~error~is~fatal.
9615 }
9616 \@@_msg_new:nn { tabularnote~forbidden }
9617 {
9618   Forbidden~command.\\
9619   You~can't~use~the~command~\token_to_str:N\tabularnote\
9620   ~here.~This~command~is~available~only~in~
9621   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9622   the~argument~of~a~command~\token_to_str:N \caption\ included~
9623   in~an~environment~{table}. \\
9624   This~command~will~be~ignored.
9625 }
9626 \@@_msg_new:nn { borders~forbidden }
9627 {
9628   Forbidden~key.\\
9629   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9630   because~the~option~'rounded~corners'~
9631   is~in~force~with~a~non~zero~value.\\
9632   This~key~will~be~ignored.
9633 }
9634 \@@_msg_new:nn { bottomrule~without~booktabs }
9635 {
9636   booktabs~not~loaded.\\
9637   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9638   loaded~'booktabs'.\\
9639   This~key~will~be~ignored.
9640 }
9641 \@@_msg_new:nn { enumitem~not~loaded }
9642 {
9643   enumitem~not~loaded.\\
9644   You~can't~use~the~command~\token_to_str:N\tabularnote\
9645   ~because~you~haven't~loaded~'enumitem'.\\
9646   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9647   ignored~in~the~document.
9648 }
9649 \@@_msg_new:nn { tikz~without~tikz }
9650 {
9651   Tikz~not~loaded.\\
9652   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9653   loaded.~If~you~go~on,~that~key~will~be~ignored.
9654 }
9655 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9656 {
9657   Tikz~not~loaded.\\
9658   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9659   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~

```

```

9660     You-can-go-on-but-you-will-have-another-error-if-you-actually-
9661     use-that-custom-line.
9662 }
9663 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9664 {
9665     Tikz-not-loaded.\
9666     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9667     command~'\token_to_str:N\Block')~but-tikz-is-not-loaded.~
9668     That-key-will-be-ignored.
9669 }
9670 \@@_msg_new:nn { without-color-inside }
9671 {
9672     If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9673     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9674     outside~\token_to_str:N \CodeBefore,~you~
9675     should-have-used-the-key~'color-inside'~in-your~\@@_full_name_env:.\
9676     You-can-go-on-but-you-may-need-more-compilations.
9677 }
9678 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9679 {
9680     Erroneous-use.\
9681     In-a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
9682     which-is-forbidden~(you-should-use~'color'~inside-the-key~'tikz').~
9683     The-key~'color'~will-be-discarded.
9684 }
9685 \@@_msg_new:nn { Wrong-last-row }
9686 {
9687     Wrong-number.\
9688     You-have-used~'last-row=~\int_use:N \l_@@_last_row_int'~but-your~
9689     \@@_full_name_env:\ seems-to-have~\int_use:N \c@iRow \ rows.~
9690     If-you-go-on,~the-value-of~\int_use:N \c@iRow \ will-be-used-for~
9691     last-row.~You-can-avoid-this-problem-by-using~'last-row'~
9692     without-value~(more-compilations-might-be-necessary).
9693 }
9694 \@@_msg_new:nn { Yet-in-env }
9695 {
9696     Nested-environments.\
9697     Environments-of-nicematrix-can't-be-nested.\
9698     This-error-is-fatal.
9699 }
9700 \@@_msg_new:nn { Outside-math-mode }
9701 {
9702     Outside-math-mode.\
9703     The~\@@_full_name_env:\ can-be-used-only-in-math-mode~
9704     (and-not-in~\token_to_str:N \vcenter).\
9705     This-error-is-fatal.
9706 }
9707 \@@_msg_new:nn { One-letter-allowed }
9708 {
9709     Bad-name.\
9710     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\
9711     It-will-be-ignored.
9712 }
9713 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9714 {
9715     Environment~{TabularNote}~forbidden.\
9716     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9717     but~*before*~the~\token_to_str:N \CodeAfter.\
9718     This-environment~{TabularNote}~will-be-ignored.
9719 }

```



```

9720 \@@_msg_new:nn { varwidth~not~loaded }
9721 {
9722   varwidth~not~loaded.\\
9723   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9724   loaded.\\
9725   Your~column~will~behave~like~'p'.
9726 }
9727 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9728 {
9729   Unknow~key.\\
9730   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9731   \c_@@_available_keys_str
9732 }
9733 {
9734   The~available~keys~are~(in~alphabetic~order):~
9735   color,~
9736   dotted,~
9737   multiplicity,~
9738   sep~color,~
9739   tikz,~and~total~width.
9740 }
9741
9742 \@@_msg_new:nnn { Unknown~key~for~Block }
9743 {
9744   Unknown~key.\\
9745   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9746   \Block.\\ It~will~be~ignored. \\
9747   \c_@@_available_keys_str
9748 }
9749 {
9750   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9751   hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9752   respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9753 }
9754 \@@_msg_new:nnn { Unknown~key~for~Brace }
9755 {
9756   Unknown~key.\\
9757   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9758   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9759   It~will~be~ignored. \\
9760   \c_@@_available_keys_str
9761 }
9762 {
9763   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9764   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9765   right~shorten)~and~yshift.
9766 }
9767 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9768 {
9769   Unknown~key.\\
9770   The~key~'\l_keys_key_str'~is~unknown.\\
9771   It~will~be~ignored. \\
9772   \c_@@_available_keys_str
9773 }
9774 {
9775   The~available~keys~are~(in~alphabetic~order):~
9776   delimiters/color,~
9777   rules~(with~the~subkeys~'color'~and~'width'),~
9778   sub~matrix~(several~subkeys)~
9779   and~xdots~(several~subkeys).~
9780   The~latter~is~for~the~command~\token_to_str:N \line.
9781 }

```

```

9782 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9783 {
9784   Unknown~key.\\
9785   The~key~'\l_keys_key_str'~is~unknown.\\
9786   It~will~be~ignored. \\
9787   \c_@@_available_keys_str
9788 }
9789 {
9790   The~available~keys~are~(in~alphabetic~order):~
9791   create~cell~nodes,~
9792   delimiters/color~and~
9793   sub~matrix~(several~subkeys).
9794 }
9795 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9796 {
9797   Unknown~key.\\
9798   The~key~'\l_keys_key_str'~is~unknown.\\
9799   That~key~will~be~ignored. \\
9800   \c_@@_available_keys_str
9801 }
9802 {
9803   The~available~keys~are~(in~alphabetic~order):~
9804   'delimiters/color',~
9805   'extra~height',~
9806   'hlines',~
9807   'hvlines',~
9808   'left~xshift',~
9809   'name',~
9810   'right~xshift',~
9811   'rules'~(with~the~subkeys~'color'~and~'width'),~
9812   'slim',~
9813   'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
9814   and~'right~xshift').\\
9815 }
9816 \@@_msg_new:nnn { Unknown~key~for~notes }
9817 {
9818   Unknown~key.\\
9819   The~key~'\l_keys_key_str'~is~unknown.\\
9820   That~key~will~be~ignored. \\
9821   \c_@@_available_keys_str
9822 }
9823 {
9824   The~available~keys~are~(in~alphabetic~order):~
9825   bottomrule,~
9826   code~after,~
9827   code~before,~
9828   detect~duplicates,~
9829   enumitem~keys,~
9830   enumitem~keys~para,~
9831   para,~
9832   label~in~list,~
9833   label~in~tabular~and~
9834   style.
9835 }
9836 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9837 {
9838   Unknown~key.\\
9839   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9840   \token_to_str:N \RowStyle. \\
9841   That~key~will~be~ignored. \\
9842   \c_@@_available_keys_str
9843 }
9844 {

```

```

9845 The~available~keys~are~(in~alphabetic~order):~
9846 'bold',~
9847 'cell-space-top-limit',~
9848 'cell-space-bottom-limit',~
9849 'cell-space-limits',~
9850 'color',~
9851 'nb-rows'~and~
9852 'rowcolor'.
9853 }
9854 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9855 {
9856   Unknown~key.\\
9857   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9858   \token_to_str:N \NiceMatrixOptions. \\
9859   That~key~will~be~ignored. \\
9860   \c_@@_available_keys_str
9861 }
9862 {
9863   The~available~keys~are~(in~alphabetic~order):~
9864   allow-duplicate-names,~
9865   caption-above,~
9866   cell-space-bottom-limit,~
9867   cell-space-limits,~
9868   cell-space-top-limit,~
9869   code-for-first-col,~
9870   code-for-first-row,~
9871   code-for-last-col,~
9872   code-for-last-row,~
9873   corners,~
9874   custom-key,~
9875   create-extra-nodes,~
9876   create-medium-nodes,~
9877   create-large-nodes,~
9878   delimiters~(several~subkeys),~
9879   end-of-row,~
9880   first-col,~
9881   first-row,~
9882   hlines,~
9883   hvlines,~
9884   hvlines-except-borders,~
9885   last-col,~
9886   last-row,~
9887   left-margin,~
9888   light-syntax,~
9889   light-syntax-expanded,~
9890   matrix/columns-type,~
9891   no-cell-nodes,~
9892   notes~(several~subkeys),~
9893   nullify-dots,~
9894   pgf-node-code,~
9895   renew-dots,~
9896   renew-matrix,~
9897   respect-arraystretch,~
9898   rounded-corners,~
9899   right-margin,~
9900   rules~(with~the~subkeys~'color'~and~'width'),~
9901   small,~
9902   sub-matrix~(several~subkeys),~
9903   vlines,~
9904   xdots~(several~subkeys).
9905 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

9906 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9907 {
9908   Unknown~key.\\
9909   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9910   \{NiceArray\}. \\
9911   That~key~will~be~ignored. \\
9912   \c_@@_available_keys_str
9913 }
9914 {
9915   The~available~keys~are~(in~alphabetic~order):~
9916   b,~
9917   baseline,~
9918   c,~
9919   cell-space-bottom-limit,~
9920   cell-space-limits,~
9921   cell-space-top-limit,~
9922   code-after,~
9923   code-for-first-col,~
9924   code-for-first-row,~
9925   code-for-last-col,~
9926   code-for-last-row,~
9927   color-inside,~
9928   columns-width,~
9929   corners,~
9930   create-extra-nodes,~
9931   create-medium-nodes,~
9932   create-large-nodes,~
9933   extra-left-margin,~
9934   extra-right-margin,~
9935   first-col,~
9936   first-row,~
9937   hlines,~
9938   hvlines,~
9939   hvlines-except-borders,~
9940   last-col,~
9941   last-row,~
9942   left-margin,~
9943   light-syntax,~
9944   light-syntax-expanded,~
9945   name,~
9946   no-cell-nodes,~
9947   nullify-dots,~
9948   pgf-node-code,~
9949   renew-dots,~
9950   respect-arraystretch,~
9951   right-margin,~
9952   rounded-corners,~
9953   rules~(with~the~subkeys~'color'~and~'width'),~
9954   small,~
9955   t,~
9956   vlides,~
9957   xdots/color,~
9958   xdots/shorten-start,~
9959   xdots/shorten-end,~
9960   xdots/shorten-and~
9961   xdots/line-style.
9962 }

```

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).

```

9963 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9964 {
9965   Unknown~key.\\
9966   The~key~'\l_keys_key_str'~is~unknown~for~the~

```

```

9967 \@@_full_name_env:. \
9968 That~key~will~be~ignored. \
9969 \c_@@_available_keys_str
9970 }
9971 {
9972   The~available~keys~are~(in~alphabetic~order):~
9973   b,~
9974   baseline,~
9975   c,~
9976   cell-space-bottom-limit,~
9977   cell-space-limits,~
9978   cell-space-top-limit,~
9979   code-after,~
9980   code-for-first-col,~
9981   code-for-first-row,~
9982   code-for-last-col,~
9983   code-for-last-row,~
9984   color-inside,~
9985   columns-type,~
9986   columns-width,~
9987   corners,~
9988   create-extra-nodes,~
9989   create-medium-nodes,~
9990   create-large-nodes,~
9991   extra-left-margin,~
9992   extra-right-margin,~
9993   first-col,~
9994   first-row,~
9995   hlines,~
9996   hvlines,~
9997   hvlines-except-borders,~
9998   l,~
9999   last-col,~
10000   last-row,~
10001   left-margin,~
10002   light-syntax,~
10003   light-syntax-expanded,~
10004   name,~
10005   no-cell-nodes,~
10006   nullify-dots,~
10007   pgf-node-code,~
10008   r,~
10009   renew-dots,~
10010   respect-arraystretch,~
10011   right-margin,~
10012   rounded-corners,~
10013   rules~(with~the~subkeys~'color'~and~'width'),~
10014   small,~
10015   t,~
10016   vlines,~
10017   xdots/color,~
10018   xdots/shorten-start,~
10019   xdots/shorten-end,~
10020   xdots/shorten-and~
10021   xdots/line-style.
10022 }
10023 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10024 {
10025   Unknown~key.\
10026   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10027   \{NiceTabular\}. \
10028   That~key~will~be~ignored. \
10029   \c_@@_available_keys_str

```

```

10030 }
10031 {
10032   The-available-keys-are~(in~alphabetic~order):~
10033   b,~
10034   baseline,~
10035   c,~
10036   caption,~
10037   cell-space-bottom-limit,~
10038   cell-space-limits,~
10039   cell-space-top-limit,~
10040   code-after,~
10041   code-for-first-col,~
10042   code-for-first-row,~
10043   code-for-last-col,~
10044   code-for-last-row,~
10045   color-inside,~
10046   columns-width,~
10047   corners,~
10048   custom-line,~
10049   create-extra-nodes,~
10050   create-medium-nodes,~
10051   create-large-nodes,~
10052   extra-left-margin,~
10053   extra-right-margin,~
10054   first-col,~
10055   first-row,~
10056   hlines,~
10057   hvlines,~
10058   hvlines-except-borders,~
10059   label,~
10060   last-col,~
10061   last-row,~
10062   left-margin,~
10063   light-syntax,~
10064   light-syntax-expanded,~
10065   name,~
10066   no-cell-nodes,~
10067   notes~(several~subkeys),~
10068   nullify-dots,~
10069   pgf-node-code,~
10070   renew-dots,~
10071   respect-arraystretch,~
10072   right-margin,~
10073   rounded-corners,~
10074   rules~(with~the~subkeys~'color'~and~'width'),~
10075   short-caption,~
10076   t,~
10077   tabularnote,~
10078   vlines,~
10079   xdots/color,~
10080   xdots/shorten-start,~
10081   xdots/shorten-end,~
10082   xdots/shorten-and~
10083   xdots/line-style.
10084 }
10085 \@@_msg_new:nnn { Duplicate-name }
10086 {
10087   Duplicate-name.\
10088   The-name-'\l_keys_value_tl'-is-already-used-and-you-shouldn't-use~
10089   the-same-environment-name-twice.~You-can-go-on,~but,~
10090   maybe,~you-will-have-incorrect-results-especially~
10091   if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
10092   message-again,~use-the-key~'allow-duplicate-names'~in~

```

```

10093     '\token_to_str:N \NiceMatrixOptions'.\\
10094     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10095     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10096   }
10097   {
10098     The~names~already~defined~in~this~document~are:~
10099     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10100   }
10101   \@@_msg_new:nn { Option~auto~for~columns~width }
10102   {
10103     Erroneous~use.\\
10104     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10105     That~key~will~be~ignored.
10106   }
10107   \@@_msg_new:nn { NiceTabularX~without~X }
10108   {
10109     NiceTabularX~without~X.\\
10110     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10111     However,~you~can~go~on.
10112   }
10113   \@@_msg_new:nn { Preamble~forgotten }
10114   {
10115     Preamble~forgotten.\\
10116     You~have~probably~forgotten~the~preamble~of~your~
10117     \@@_full_name_env:. \\
10118     This~error~is~fatal.
10119   }

```

# Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	4
5	Parameters	9
6	The command <code>\tabularnote</code>	19
7	Command for creation of rectangle nodes	24
8	The options	25
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	73
14	The environment <code>{NiceMatrix}</code> and its variants	91
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
16	After the construction of the array	93
17	We draw the dotted lines	99
18	The actual instructions for drawing the dotted lines with Tikz	113
19	User commands available in the new environments	118
20	The command <code>\line</code> accessible in code-after	124
21	The command <code>\RowStyle</code>	126
22	Colors of cells, rows and columns	129
23	The vertical and horizontal rules	141
24	The empty corners	156
25	The environment <code>{NiceMatrixBlock}</code>	158
26	The extra nodes	159
27	The blocks	164
28	How to draw the dotted lines transparently	184
29	Automatic arrays	184
30	The redefinition of the command <code>\dotfill</code>	185



31	The command <code>\diagbox</code>	186
32	The keyword <code>\CodeAfter</code>	187
33	The delimiters in the preamble	188
34	The command <code>\SubMatrix</code>	189
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	197
36	The command <code>TikzEveryCell</code>	200
37	The command <code>\ShowCellNames</code>	202
38	We process the options at package loading	205
39	About the package underscore	206
40	Error messages of the package	207