

# The code of the package `nicematrix`\*

F. Pantigny  
fpantigny@wanadoo.fr

March 12, 2025

## 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 \ProvidesExplPackage
4   {nicematrix}
5   {\myfiledate}
6   {\myfileversion}
7   {Enhanced arrays with the help of PGF/TikZ}
8 \msg_new:nnn { nicematrix } { latex-too-old }
9   {
10    Your-LaTeX-release-is-too-old. \
11    You-need-at-least-a-the-version-of-2023-11-01
12  }
13 \providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
14 \IfFormatAtLeastTF
15   { 2023-11-01 }
16   { }
17   { \msg_fatal:nn { nicematrix } { latex-too-old } }
18 \ProvideDocumentCommand{\IfPackageLoadedT}{mm}
19   {\IfPackageLoadedTF{#1}{#2}{}}
20
21 \ProvideDocumentCommand{\IfPackageLoadedF}{mm}
22   {\IfPackageLoadedTF{#1}{#2}{}}
```

---

\*This document corresponds to the version 7.1a of `nicematrix`, at the date of 2025/03/04.

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

```
23 \RequirePackage { amsmath }
```

```
24 \RequirePackage { array }
```

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

```
25 \bool_const:Nn \c_@@_recent_array_bool
26 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
27 \bool_const:Nn \c_@@_testphase_table_bool
28 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

```
29 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
30 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
31 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
32 \cs_generate_variant:Nn \@@_error:nn { n e }
33 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
34 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
35 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
36 \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).

```
37 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
38 {
39   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
40     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
41     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
42 }
```

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

```
43 \cs_new_protected:Npn \@@_error_or_warning:n
44 { \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”.

```
45 \bool_new:N \g_@@_messages_for_Overleaf_bool
46 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
47 {
48   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
49   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
50 }
```

```
51 \cs_new_protected:Npn \@@_msg_redirect_name:nn
52 { \msg_redirect_name:nnn { nicematrix } }
53 \cs_new_protected:Npn \@@_gredirect_none:n #1
54 {
55   \group_begin:
56   \globaldefs = 1
57   \@@_msg_redirect_name:nn { #1 } { none }
58   \group_end:
59 }
60 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
61 {
62   \@@_error:n { #1 }
63   \@@_gredirect_none:n { #1 }
```

```

64 }
65 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
66 {
67   \@@_warning:n { #1 }
68   \@@_gredirect_none:n { #1 }
69 }

```

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

```

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

72 \@@_msg_new:nn { mdwtab~loaded }
73 {
74   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
75   This~error~is~fatal.
76 }

77 \hook_gput_code:nnn { begindocument / end } { . }
78 { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab~loaded } } }

```

## 2 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.

*Example :*

`\@@_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`).

```

79 \cs_new_protected:Npn \@@_collect_options:n #1
80 {
81   \peek_meaning:NTF [
82     { \@@_collect_options:nw { #1 } }
83     { #1 { } }
84 }

```

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

```

85 \NewDocumentCommand \@@_collect_options:nw { m r[] }
86 { \@@_collect_options:nn { #1 } { #2 } }
87
88 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
89 {
90   \peek_meaning:NTF [
91     { \@@_collect_options:nnw { #1 } { #2 } }
92     { #1 { #2 } }
93 }
94
95 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
96 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

### 3 Technical definitions

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

```

97 \tl_const:Nn \c_@@_b_tl { b }
98 \tl_const:Nn \c_@@_c_tl { c }
99 \tl_const:Nn \c_@@_l_tl { l }
100 \tl_const:Nn \c_@@_r_tl { r }
101 \tl_const:Nn \c_@@_all_tl { all }
102 \tl_const:Nn \c_@@_dot_tl { . }
103 \str_const:Nn \c_@@_r_str { r }
104 \str_const:Nn \c_@@_c_str { c }
105 \str_const:Nn \c_@@_l_str { l }

```

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).

```

106 \tl_new:N \l_@@_argspec_tl

107 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
108 \cs_generate_variant:Nn \str_lowercase:n { o }
109 \cs_generate_variant:Nn \str_set:Nn { N o }
110 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
111 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
112 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
113 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
114 \cs_generate_variant:Nn \dim_min:nn { v }
115 \cs_generate_variant:Nn \dim_max:nn { v }

116 \hook_gput_code:nnn { begindocument } { . }
117 {
118   \IfPackageLoadedTF { tikz }
119   {

```

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.

```

120 \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
121 \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
122 }
123 {
124   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
125   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
126 }
127 }

```

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`).

```

128 \IfClassLoadedTF { revtex4-1 }
129 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
130 {
131   \IfClassLoadedTF { revtex4-2 }
132   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
133   {

```

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

```

134     \cs_if_exist:NT \rvtx@ifformat@geq
135     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
136     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
137   }
138 }

```

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.

```

139 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
140 {
141   \iow_now:Nn \@mainaux
142   {
143     \ExplSyntaxOn
144     \cs_if_free:NT \pgfsyspdfmark
145     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
146     \ExplSyntaxOff
147   }
148   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
149 }

```

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

```

150 \ProvideDocumentCommand \iddots { }
151 {
152   \mathinner
153   {
154     \tex_mkern:D 1 mu
155     \box_move_up:nn { 1 pt } { \hbox { . } }
156     \tex_mkern:D 2 mu
157     \box_move_up:nn { 4 pt } { \hbox { . } }
158     \tex_mkern:D 2 mu
159     \box_move_up:nn { 7 pt }
160     { \vbox:n { \kern 7 pt \hbox { . } } }
161     \tex_mkern:D 1 mu
162   }
163 }

```

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.

```

164 \hook_gput_code:nnn { begindocument } { . }
165 {
166   \IfPackageLoadedT { booktabs }
167   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
168 }
169 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
170 {
171   \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun

```

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

```

172   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
173   {

```

`\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

174     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
175     { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
176   }
177 }

```

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

```

178 \hook_gput_code:nnn { begindocument } { . }
179 {
180   \IfPackageLoadedF { colortbl }
181   {

```

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.

```

182     \cs_set_protected:Npn \CT@arc@ { }
183     \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
184     \cs_set_nopar:Npn \CT@arc #1 #2
185     {
186       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
187       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
188     }

```

Idem for `\CT@drs@`.

```

189     \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
190     \cs_set_nopar:Npn \CT@drs #1 #2
191     {
192       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
193       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
194     }
195     \cs_set_nopar:Npn \hline
196     {
197       \noalign { \ifnum 0 = ` } \fi
198       \cs_set_eq:NN \hskip \vskip
199       \cs_set_eq:NN \vrule \hrule
200       \cs_set_eq:NN \@width \@height
201       { \CT@arc@ \vline }
202       \futurelet \reserved@a
203       \@xhline
204     }
205   }
206 }

```

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.

```

207 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
208 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
209 {
210   \int_if_zero:nT \l_@@_first_col_int { \omit & }
211   \int_compare:nNnT { #1 } > \c_one_int
212   { \multispan { \int_eval:n { #1 - 1 } } & }
213   \multispan { \int_eval:n { #2 - #1 + 1 } }
214   {
215     \CT@arc@
216     \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>

```

217     \skip_horizontal:N \c_zero_dim
218   }

```

---

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

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`.

```

219   \everycr { }
220   \cr
221   \noalign { \skip_vertical:N -\arrayrulewidth }
222 }

```

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.

```

223 \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`.

```

224 { \@@_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*.

```

225 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
226 \cs_generate_variant:Nn \@@_cline_i:nn { e }
227 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
228 {
229   \tl_if_empty:nTF { #3 }
230   { \@@_cline_iii:w #1|#2-#2 \q_stop }
231   { \@@_cline_ii:w #1|#2-#3 \q_stop }
232 }
233 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
234 { \@@_cline_iii:w #1|#2-#3 \q_stop }
235 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
236 {

```

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).

```

237   \int_compare:nNnT { #1 } < { #2 }
238   { \multispan { \int_eval:n { #2 - #1 } } & }
239   \multispan { \int_eval:n { #3 - #2 + 1 } }
240   {
241     \CT@arc@
242     \leaders \hrule \@height \arrayrulewidth \hfill
243     \skip_horizontal:N \c_zero_dim
244   }

```

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

```

245   \peek_meaning_remove_ignore_spaces:NNTF \cline
246   { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
247   { \everycr { } \cr }
248 }

```

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

```

249 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

250 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
251 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
252 {
253   \tl_if_blank:nF { #1 }
254   {
255     \tl_if_head_eq_meaning:nNTF { #1 } [
256       { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
257       { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
258     ]
259   }

```

```

260 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
261 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
262 {
263   \tl_if_head_eq_meaning:nNTF { #1 } [
264     { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
265     { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
266   ]

```

The following command must *not* be protected since it will be used to write instructions in the `\g_@@_pre_code_before_tl`.

```

267 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
268 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
269 {
270   \tl_if_head_eq_meaning:nNTF { #2 } [
271     { #1 #2 }
272     { #1 { #2 } }
273   ]

```

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

```

274 \cs_generate_variant:Nn \@@_color:n { o }
275 \cs_new_protected:Npn \@@_color:n #1
276 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

```

```

277 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
278 {
279   \tl_set_rescan:Nno
280     #1
281     {
282       \char_set_catcode_other:N >
283       \char_set_catcode_other:N <
284     }
285   #1
286 }

```

## 4 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.

```

287 \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).

```

288 \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.

```

289 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
290 { \int_use:N \g_@@_env_int }

```

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

```

291 \cs_new_protected:Npn \@@_qpoint:n #1
292 { \pgfpointanchor { \@@_env: - #1 } { center } }

```



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

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

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

```
294 \bool_new:N \g_@@_delims_bool
295 \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.

```
296 \bool_new:N \l_@@_preamble_bool
297 \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.

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

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

```
299 \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.

```
300 \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).

```
301 \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.).

```
302 \dim_new:N \l_@@_col_width_dim
303 \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.

```
304 \int_new:N \g_@@_row_total_int
305 \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).

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

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

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

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

```
308 \tl_new:N \l_@@_hpos_cell_tl
309 \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`.

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

Idem for the mono-row blocks.

```
311 \dim_new:N \g_@@_blocks_ht_dim
312 \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}`).

```
313 \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`.

```
314 \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.

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

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

```
316 \bool_new:N \l_@@_notes_detect_duplicates_bool
317 \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.

```
318 \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).

```
319 \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).

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

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

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

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

```
322 \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 (the `X` columns of `nicematrix` are inspired by those of `tabularx`).

```
323 \bool_new:N \l_@@_X_bool
324 \bool_new:N \g_@@_caption_finished_bool
```

The following boolean will be raised when the key `no-cell-nodes` is used.

```
325 \bool_new:N \l_@@_no_cell_nodes_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 }`).

```
326 \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.

```
327 \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.

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

```
329 \tl_new:N \g_@@_left_delim_tl
```

```
330 \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}`).

```
331 \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`).

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

For `\multicolumn`.

```
333 \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`.

```
334 \tl_new:N \l_@@_columns_type_tl
```

```
335 \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 `:`.

```
336 \tl_new:N \l_@@_xdots_down_tl
```

```
337 \tl_new:N \l_@@_xdots_up_tl
```

```
338 \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`).

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

```
340 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
341 {
```

```
342   \if_mode_math: \else:
```

```
343     \@@_fatal:n { Outside~math~mode }
```

```
344   \fi:
```

```
345 }
```

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.

```
346 \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”.

```
347 \colorlet { nicematrix-last-col } { . }
```

```
348 \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`).

```
349 \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`.

```
350 \tl_new:N \g_@@_com_or_env_str
351 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
352 \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:eeTF` and not `\tl_if_eq:eeTF` because we need to be fully expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
353 \cs_new:Npn \@@_full_name_env:
354 {
355   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
356   { command \space \c_backslash_str \g_@@_name_env_str }
357   { environment \space \{ \g_@@_name_env_str \} }
358 }
```

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

```
359 \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.

```
360 \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.

```
361 \tl_new:N \g_@@_pre_code_before_tl
362 \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.

```
363 \tl_new:N \g_@@_pre_code_after_tl
364 \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.

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

The following parameter will be raised when a block contains an ampersand (`&`) in its content (`=label`).

```
366 \bool_new:N \l_@@_ampersand_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.

```
367 \int_new:N \l_@@_old_iRow_int
368 \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).

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

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

```
370 \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.

```
371 \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.

```
372 \bool_new:N \l_@@_X_columns_aux_bool
```

```
373 \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.

```
374 \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).

```
375 \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).

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

```
377 \tl_new:N \l_@@_code_before_tl
```

```
378 \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).

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

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

```
380 \dim_new:N \l_@@_x_initial_dim
```

```
381 \dim_new:N \l_@@_y_initial_dim
```

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

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

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

```
384 \dim_new:N \l_@@_tmpc_dim
```

```
385 \dim_new:N \l_@@_tmpd_dim
```

```
386 \dim_new:N \l_@@_tmpe_dim
```

```
387 \dim_new:N \l_@@_tmpf_dim
```

```

388 \dim_new:N \g_@@_dp_row_zero_dim
389 \dim_new:N \g_@@_ht_row_zero_dim
390 \dim_new:N \g_@@_ht_row_one_dim
391 \dim_new:N \g_@@_dp_ante_last_row_dim
392 \dim_new:N \g_@@_ht_last_row_dim
393 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

394 \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”.

```

395 \dim_new:N \g_@@_width_last_col_dim
396 \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.

```

397 \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!).

```

398 \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).

In the `\CodeBefore`, the value of `\g_@@_pos_of_blocks_seq` will be the value read in the `aux` file from a previous run. However, in the `\CodeBefore`, the commands `\EmptyColumn` and `\EmptyRow` will write virtual positions of blocks in the following sequence.

```

399 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

The, after the execution of the `\CodeBefore`, the sequence `\g_@@_pos_of_blocs_seq` will be erased and replaced by the value of `\g_@@_future_pos_of_blocks_seq`.

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}`.

```

400 \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.

```

401 \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 list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

402 \clist_new:N \l_@@_corners_cells_clist

```

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`).

```

403 \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.

```
404 \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).

```
405 \seq_new:N \g_@@_multicolumn_cells_seq
406 \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).

```
407 \int_new:N \l_@@_row_min_int
408 \int_new:N \l_@@_row_max_int
409 \int_new:N \l_@@_col_min_int
410 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
411 \int_new:N \l_@@_start_int
412 \int_set_eq:NN \l_@@_start_int \c_one_int
413 \int_new:N \l_@@_end_int
414 \int_new:N \l_@@_local_start_int
415 \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.

```
416 \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).

```
417 \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`.

```
418 \tl_new:N \l_@@_fill_tl
419 \tl_new:N \l_@@_opacity_tl
420 \tl_new:N \l_@@_draw_tl
421 \seq_new:N \l_@@_tikz_seq
422 \clist_new:N \l_@@_borders_clist
423 \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.

```
424 \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`.

```
425 \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.

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

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

```
427 \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`).

```
428 \str_new:N \l_@@_hpos_block_str
429 \str_set:Nn \l_@@_hpos_block_str { c }
430 \bool_new:N \l_@@_hpos_of_block_cap_bool
431 \bool_new:N \l_@@_p_block_bool
```

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

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

For the vertical position, the possible values are `c`, `t`, `b`, `T` and `B` (but `\l_@@_vpos_block_str` will remain empty if the user doesn’t use a key for the vertical position).

```
433 \str_new:N \l_@@_vpos_block_str
```

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

```
434 \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).

```
435 \bool_new:N \l_@@_vlines_block_bool
436 \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.

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

438 \dim_new:N \l_@@_submatrix_extra_height_dim
439 \dim_new:N \l_@@_submatrix_left_xshift_dim
440 \dim_new:N \l_@@_submatrix_right_xshift_dim
441 \clist_new:N \l_@@_hlines_clist
442 \clist_new:N \l_@@_vlines_clist
443 \clist_new:N \l_@@_submatrix_hlines_clist
444 \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}`).

```
445 \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.

```
446 \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`).

```
447 \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.

```
448 \int_new:N \l_@@_first_row_int
449 \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.

```
450 \int_new:N \l_@@_first_col_int
451 \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).

```
452 \int_new:N \l_@@_last_row_int
453 \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>

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

Idem for `\l_@@_last_col_without_value_bool`

```
455 \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.

```
456 \int_new:N \l_@@_last_col_int
457 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

---

<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.

```

\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.

```

458 \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`).

```

459 \bool_new:N \l_@@_in_last_col_bool

```

## Some utilities

```

460 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
461 {

```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```

462 \cs_set_nopar:Npn \l_tmpa_tl { #1 }
463 \cs_set_nopar:Npn \l_tmpb_tl { #2 }
464 }

```

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.

```

465 \cs_new_protected:Npn \@@_expand_clist:N #1
466 {
467 \clist_if_in:NnF #1 { all }
468 {
469 \clist_clear:N \l_tmpa_clist
470 \clist_map_inline:Nn #1
471 {

```

We recall that `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```

472 \tl_if_in:nnTF { ##1 } { - }
473 { \@@_cut_on_hyphen:w ##1 \q_stop }
474 {

```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```

475 \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
476 \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
477 }
478 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
479 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
480 }
481 \tl_set_eq:NN #1 \l_tmpa_clist
482 }
483 }

```

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.

```

484 \hook_gput_code:nnn { begindocument } { . }
485 {
486 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
487 \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
488 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
489 }

```

## 5 The command `\tabularnote`

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

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

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

```
490 \newcounter { tabularnote }
```

We want to avoid error messages for duplicate labels when the package `hyperref` is used. That's why we will count all the tabular notes of the whole document with `\g_@@_tabularnote_int`.

```
491 \int_new:N \g_@@_tabularnote_int
492 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
493 \seq_new:N \g_@@_notes_seq
494 \seq_new:N \g_@@_notes_in_caption_seq
```

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

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

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

```
496 \seq_new:N \l_@@_notes_labels_seq
497 \newcounter { nicematrix_draft }
```

---

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

```

498 \cs_new_protected:Npn \@@_notes_format:n #1
499 {
500   \setcounter { nicematrix_draft } { #1 }
501   \@@_notes_style:n { nicematrix_draft }
502 }

```

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

```

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

```

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

```

504 \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`.

```

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

```

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

```

506 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }

```

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

```

507 \hook_gput_code:nnn { begindocument } { . }
508 {
509   \IfPackageLoadedTF { enumitem }
510   {

```

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.

```

511     \newlist { tabularnotes } { enumerate } { 1 }
512     \setlist [ tabularnotes ]
513     {
514       topsep = 0pt ,
515       noitemsep ,
516       leftmargin = * ,
517       align = left ,
518       labelsep = 0pt ,
519       label =
520       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
521     }
522     \newlist { tabularnotes* } { enumerate* } { 1 }
523     \setlist [ tabularnotes* ]
524     {
525       afterlabel = \nobreak ,
526       itemjoin = \quad ,
527       label =
528       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
529     }

```

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`.

```

530     \NewDocumentCommand \tabularnote { o m }
531     {
532       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } \l_@@_in_env_bool

```

```

533     {
534         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
535         { \@@_error:n { tabularnote~forbidden } }
536     {
537         \bool_if:NTF \l_@@_in_caption_bool
538         \@@_tabularnote_caption:nn
539         \@@_tabularnote:nn
540         { #1 } { #2 }
541     }
542 }
543 }
544 }
545 {
546     \NewDocumentCommand \tabularnote { o m }
547     {
548         \@@_error_or_warning:n { enumitem~not~loaded }
549         \@@_gredirect_none:n { enumitem~not~loaded }
550     }
551 }
552 }
553 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
554 { \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`.

```

555 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
556 {

```

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).

```

557     \int_zero:N \l_tmpa_int
558     \bool_if:NT \l_@@_notes_detect_duplicates_bool
559     {

```

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`.

```

560     \int_zero:N \l_tmpb_int
561     \seq_map_indexed_inline:Nn \g_@@_notes_seq
562     {
563         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
564         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
565         {
566             \tl_if_novalue:nTF { #1 }
567             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
568             { \int_set:Nn \l_tmpa_int { ##1 } }
569             \seq_map_break:
570         }
571     }
572     \int_if_zero:nF \l_tmpa_int
573     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
574 }
575 \int_if_zero:nT \l_tmpa_int
576 {

```

```

577     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
578     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
579   }
580   \seq_put_right:Ne \l_@@_notes_labels_seq
581   {
582     \tl_if_novalue:nTF { #1 }
583     {
584       \@@_notes_format:n
585       {
586         \int_eval:n
587         {
588           \int_if_zero:nTF \l_tmpa_int
589             \c@tabularnote
590             \l_tmpa_int
591         }
592       }
593     }
594     { #1 }
595   }
596   \peek_meaning:NF \tabularnote
597   {

```

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`.

```

598     \hbox_set:Nn \l_tmpa_box
599     {

```

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

```

600     \@@_notes_label_in_tabular:n
601     {
602       \seq_use:Nnnn
603       \l_@@_notes_labels_seq { , } { , } { , }
604     }
605   }

```

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

```

606     \int_gdecr:N \c@tabularnote
607     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

The following line is only to avoid error messages for multiply defined labels when the package `hyperref` is used.

```

608     \int_gincr:N \g_@@_tabularnote_int
609     \refstepcounter { tabularnote }
610     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
611       { \int_gincr:N \c@tabularnote }
612     \seq_clear:N \l_@@_notes_labels_seq
613     \bool_lazy_or:nnTF
614       { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
615       { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
616     {
617       \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).

```

618     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
619   }
620   { \box_use:N \l_tmpa_box }
621 }
622 }

```

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`.

```

623 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
624 {
625   \bool_if:NTF \g_@@_caption_finished_bool
626   {
627     \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
628     { \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`!

```

629     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
630     { \@@_error:n { Identical-notes-in-caption } }
631   }
632 {

```

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

```

633     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
634     {

```

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.

```

635         \bool_gset_true:N \g_@@_caption_finished_bool
636         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
637         \int_gzero:N \c@tabularnote
638     }
639     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
640 }

```

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!

```

641     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
642     \seq_put_right:Ne \l_@@_notes_labels_seq
643     {
644       \tl_if_novalue:nTF { #1 }
645       { \@@_notes_format:n { \int_use:N \c@tabularnote } }
646       { #1 }
647     }
648     \peek_meaning:NF \tabularnote
649     {
650       \@@_notes_label_in_tabular:n
651       { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
652       \seq_clear:N \l_@@_notes_labels_seq
653     }
654 }

655 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
656 { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

## 6 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.

```

657 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
658 {
659   \begin { pgfscope }
660   \pgfset
661   {
662     inner~sep = \c_zero_dim ,
663     minimum~size = \c_zero_dim
664   }
665   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
666   \pgfnode
667   { rectangle }
668   { center }
669   {
670     \vbox_to_ht:nn
671     { \dim_abs:n { #5 - #3 } }
672     {
673       \vfill
674       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
675     }
676   }
677   { #1 }
678   { }
679   \end { pgfscope }
680 }

```

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.

```

681 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
682 {
683   \begin { pgfscope }
684   \pgfset
685   {
686     inner~sep = \c_zero_dim ,
687     minimum~size = \c_zero_dim
688   }
689   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
690   \pgfpointdiff { #3 } { #2 }
691   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
692   \pgfnode
693   { rectangle }
694   { center }
695   {
696     \vbox_to_ht:nn
697     { \dim_abs:n \l_tmpb_dim }
698     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
699   }
700   { #1 }
701   { }
702   \end { pgfscope }
703 }

```

## 7 The options

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

```

704 \tl_new:N \l_@@_caption_tl
705 \tl_new:N \l_@@_short_caption_tl
706 \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).

```
707 \bool_new:N \l_@@_caption_above_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`.

```
708 \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`).

```
709 \dim_new:N \l_@@_cell_space_top_limit_dim
710 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
711 \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.

```
712 \dim_new:N \l_@@_xdots_inter_dim
713 \hook_gput_code:nnn { begindocument } { . }
714 { \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).

```
715 \dim_new:N \l_@@_xdots_shorten_start_dim
716 \dim_new:N \l_@@_xdots_shorten_end_dim
717 \hook_gput_code:nnn { begindocument } { . }
718 {
719   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
720   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
721 }
```

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.

```
722 \dim_new:N \l_@@_xdots_radius_dim
723 \hook_gput_code:nnn { begindocument } { . }
724 { \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.

```
725 \tl_new:N \l_@@_xdots_line_style_tl
726 \tl_const:Nn \c_@@_standard_tl { standard }
727 \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`.

```
728 \bool_new:N \l_@@_light_syntax_bool
729 \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).

```
730 \tl_new:N \l_@@_baseline_tl
731 \tl_set:Nn \l_@@_baseline_tl { c }
```

The following parameter corresponds to the key `ampersand-in-blocks`

```
732 \bool_new:N \l_@@_amp_in_blocks_bool
```

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`).

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

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

```
734 \bool_new:N \l_@@_parallelize_diags_bool
735 \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.

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

```
737 \dim_new:N \l_@@_notes_above_space_dim
738 \hook_gput_code:nnn { begindocument } { . }
739 { \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.).

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

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

```
741 \cs_new_protected:Npn \@@_reset_arraystretch:
742 { \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).

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

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

```
744 \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.

```
745 \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”.

```
746 \bool_new:N \l_@@_medium_nodes_bool
747 \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).

```
748 \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`).

```
749 \dim_new:N \l_@@_left_margin_dim
750 \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`.

```
751 \dim_new:N \l_@@_extra_left_margin_dim
752 \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.

```
753 \tl_new:N \l_@@_end_of_row_tl
754 \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 “:”.

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

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

```
756 \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.

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

```
758 \keys_define:nn { nicematrix / xdots }
759 {
760   shorten-start .code:n =
761     \hook_gput_code:nnn { begindocument } { . }
762     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
763   shorten-end .code:n =
764     \hook_gput_code:nnn { begindocument } { . }
765     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
766   shorten-start .value_required:n = true ,
767   shorten-end .value_required:n = true ,
768   shorten .code:n =
769     \hook_gput_code:nnn { begindocument } { . }
770     {
771       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
772       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
773     } ,
774   shorten .value_required:n = true ,
775   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
776   horizontal-labels .default:n = true ,
777   line-style .code:n =
778     {
779       \bool_lazy_or:nnTF
780         { \cs_if_exist_p:N \tikzpicture }
```

```

781      { \str_if_eq_p:nn { #1 } { standard } }
782      { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
783      { \@@_error:n { bad-option-for-line-style } }
784    } ,
785    line-style .value_required:n = true ,
786    color .tl_set:N = \l_@@_xdots_color_tl ,
787    color .value_required:n = true ,
788    radius .code:n =
789      \hook_gput_code:nnn { begindocument } { . }
790      { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
791    radius .value_required:n = true ,
792    inter .code:n =
793      \hook_gput_code:nnn { begindocument } { . }
794      { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
795    radius .value_required:n = true ,

```

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

```

796    down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
797    up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
798    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).

```

799    draw-first .code:n = \prg_do_nothing: ,
800    unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
801  }

```

```

802 \keys_define:nn { nicematrix / rules }
803 {
804   color .tl_set:N = \l_@@_rules_color_tl ,
805   color .value_required:n = true ,
806   width .dim_set:N = \arrayrulewidth ,
807   width .value_required:n = true ,
808   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
809 }

```

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

```

810 \keys_define:nn { nicematrix / Global }
811 {
812   color-inside .code:n =
813     \@@_warning_gredirect_none:n { key-color-inside } ,
814   colortbl-like .code:n =
815     \@@_warning_gredirect_none:n { key-color-inside } ,
816   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
817   ampersand-in-blocks .default:n = true ,
818   &-in-blocks .meta:n = ampersand-in-blocks ,
819   no-cell-nodes .code:n =
820     \bool_set_true:N \l_@@_no_cell_nodes_bool
821     \cs_set_protected:Npn \@@_node_for_cell:
822       { \set@color \box_use_drop:N \l_@@_cell_box } ,
823   no-cell-nodes .value_forbidden:n = true ,
824   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
825   rounded-corners .default:n = 4 pt ,
826   custom-line .code:n = \@@_custom_line:n { #1 } ,
827   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
828   rules .value_required:n = true ,
829   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
830   standard-cline .default:n = true ,

```

```

831 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
832 cell-space-top-limit .value_required:n = true ,
833 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
834 cell-space-bottom-limit .value_required:n = true ,
835 cell-space-limits .meta:n =
836 {
837     cell-space-top-limit = #1 ,
838     cell-space-bottom-limit = #1 ,
839 } ,
840 cell-space-limits .value_required:n = true ,
841 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
842 light-syntax .code:n =
843     \bool_set_true:N \l_@@_light_syntax_bool
844     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
845 light-syntax .value_forbidden:n = true ,
846 light-syntax-expanded .code:n =
847     \bool_set_true:N \l_@@_light_syntax_bool
848     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
849 light-syntax-expanded .value_forbidden:n = true ,
850 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
851 end-of-row .value_required:n = true ,
852 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
853 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
854 last-row .int_set:N = \l_@@_last_row_int ,
855 last-row .default:n = -1 ,
856 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
857 code-for-first-col .value_required:n = true ,
858 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
859 code-for-last-col .value_required:n = true ,
860 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
861 code-for-first-row .value_required:n = true ,
862 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
863 code-for-last-row .value_required:n = true ,
864 hlines .clist_set:N = \l_@@_hlines_clist ,
865 vlines .clist_set:N = \l_@@_vlines_clist ,
866 hlines .default:n = all ,
867 vlines .default:n = all ,
868 vlines-in-sub-matrix .code:n =
869 {
870     \tl_if_single_token:nTF { #1 }
871     {
872         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
873         { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

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

```

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

```

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

```

917 \keys_define:nn { nicematrix / environments }
918 {
919   corners .clist_set:N = \l_@@_corners_clist ,
920   corners .default:n = { NW , SW , NE , SE } ,
921   code-before .code:n =
922     {
923       \tl_if_empty:nF { #1 }
924       {
925         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
926         \bool_set_true:N \l_@@_code_before_bool
927       }
928     } ,
929   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}`.

```

930   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
931   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
932   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
933   baseline .tl_set:N = \l_@@_baseline_tl ,
934   baseline .value_required:n = true ,
935   columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

936   \str_if_eq:eeTF { #1 } { auto }
937   { \bool_set_true:N \l_@@_auto_columns_width_bool }
938   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
939   columns-width .value_required:n = true ,
940   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.

```

941 \legacy_if:nF { measuring@ }
942 {
943   \str_set:Ne \l_tmpa_str { #1 }
944   \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
945   { \@@_error:nn { Duplicate~name } { #1 } }
946   { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
947   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
948 } ,
949 name .value_required:n = true ,
950 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
951 code-after .value_required:n = true ,
952 }
953 \keys_define:nn { nicematrix / notes }
954 {
955   para .bool_set:N = \l_@@_notes_para_bool ,
956   para .default:n = true ,
957   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
958   code-before .value_required:n = true ,
959   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
960   code-after .value_required:n = true ,
961   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
962   bottomrule .default:n = true ,
963   style .cs_set:Np = \@@_notes_style:n #1 ,
964   style .value_required:n = true ,
965   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
966   label-in-tabular .value_required:n = true ,
967   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
968   label-in-list .value_required:n = true ,
969   enumitem-keys .code:n =
970   {
971     \hook_gput_code:nnn { begindocument } { . }
972     {
973       \IfPackageLoadedT { enumitem }
974       { \setlist* [ tabularnotes ] { #1 } }
975     }
976   } ,
977   enumitem-keys .value_required:n = true ,
978   enumitem-keys-para .code:n =
979   {
980     \hook_gput_code:nnn { begindocument } { . }
981     {
982       \IfPackageLoadedT { enumitem }
983       { \setlist* [ tabularnotes* ] { #1 } }
984     }
985   } ,
986   enumitem-keys-para .value_required:n = true ,
987   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
988   detect-duplicates .default:n = true ,
989   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
990 }
991 \keys_define:nn { nicematrix / delimiters }
992 {
993   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
994   max-width .default:n = true ,
995   color .tl_set:N = \l_@@_delimiters_color_tl ,
996   color .value_required:n = true ,
997 }

```

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

```

998 \keys_define:nn { nicematrix }
999 {

```

```

1000 NiceMatrixOptions .inherit:n =
1001   { nicematrix / Global } ,
1002 NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1003 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1004 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1005 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1006 SubMatrix / rules .inherit:n = nicematrix / rules ,
1007 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1008 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1009 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1010 NiceMatrix .inherit:n =
1011   {
1012     nicematrix / Global ,
1013     nicematrix / environments ,
1014   } ,
1015 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1016 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1017 NiceTabular .inherit:n =
1018   {
1019     nicematrix / Global ,
1020     nicematrix / environments
1021   } ,
1022 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1023 NiceTabular / rules .inherit:n = nicematrix / rules ,
1024 NiceTabular / notes .inherit:n = nicematrix / notes ,
1025 NiceArray .inherit:n =
1026   {
1027     nicematrix / Global ,
1028     nicematrix / environments ,
1029   } ,
1030 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1031 NiceArray / rules .inherit:n = nicematrix / rules ,
1032 pNiceArray .inherit:n =
1033   {
1034     nicematrix / Global ,
1035     nicematrix / environments ,
1036   } ,
1037 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1038 pNiceArray / rules .inherit:n = nicematrix / rules ,
1039 }

```

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

```

1040 \keys_define:nm { nicematrix / NiceMatrixOptions }
1041 {
1042   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1043   delimiters / color .value_required:n = true ,
1044   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1045   delimiters / max-width .default:n = true ,
1046   delimiters .code:n = \keys_set:nm { nicematrix / delimiters } { #1 } ,
1047   delimiters .value_required:n = true ,
1048   width .dim_set:N = \l_@@_width_dim ,
1049   width .value_required:n = true ,
1050   last-col .code:n =
1051     \tl_if_empty:nF { #1 }
1052     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1053     \int_zero:N \l_@@_last_col_int ,
1054   small .bool_set:N = \l_@@_small_bool ,
1055   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.

```

1056 renew-matrix .code:n = \@@_renew_matrix: ,
1057 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}`.

```
1058     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.

```
1059     columns-width .code:n =
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
1060     \str_if_eq:eeTF { #1 } { auto }
1061     { \@@_error:n { Option~auto~for~columns~width } }
1062     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
1063     allow-duplicate-names .code:n =
1064     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1065     allow-duplicate-names .value_forbidden:n = true ,
1066     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1067     notes .value_required:n = true ,
1068     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1069     sub-matrix .value_required:n = true ,
1070     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1071     matrix / columns-type .value_required:n = true ,
1072     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1073     caption-above .default:n = true ,
1074     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1075 }
```

`\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.

```
1076 \NewDocumentCommand \NiceMatrixOptions { m }
1077 { \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.

```
1078 \keys_define:nn { nicematrix / NiceMatrix }
1079 {
1080     last-col .code:n = \tl_if_empty:nTF { #1 }
1081     {
1082         \bool_set_true:N \l_@@_last_col_without_value_bool
1083         \int_set:Nn \l_@@_last_col_int { -1 }
1084     }
1085     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1086     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1087     columns-type .value_required:n = true ,
1088     l .meta:n = { columns-type = l } ,
1089     r .meta:n = { columns-type = r } ,
1090     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1091     delimiters / color .value_required:n = true ,
1092     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1093     delimiters / max-width .default:n = true ,
1094     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1095     delimiters .value_required:n = true ,
1096     small .bool_set:N = \l_@@_small_bool ,
1097     small .value_forbidden:n = true ,
1098     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1099 }
```

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

```
1100 \keys_define:nn { nicematrix / NiceArray }
1101 {
```

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.

```
1102     small .bool_set:N = \l_@@_small_bool ,
1103     small .value_forbidden:n = true ,
1104     last-col .code:n = \tl_if_empty:nF { #1 }
1105         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1106         \int_zero:N \l_@@_last_col_int ,
1107     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1108     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1109     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1110 }

1111 \keys_define:nn { nicematrix / pNiceArray }
1112 {
1113     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1114     last-col .code:n = \tl_if_empty:nF { #1 }
1115         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1116         \int_zero:N \l_@@_last_col_int ,
1117     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1118     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1119     delimiters / color .value_required:n = true ,
1120     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1121     delimiters / max-width .default:n = true ,
1122     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1123     delimiters .value_required:n = true ,
1124     small .bool_set:N = \l_@@_small_bool ,
1125     small .value_forbidden:n = true ,
1126     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1127     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1128     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1129 }
```

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

```
1130 \keys_define:nn { nicematrix / NiceTabular }
1131 {
```

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.

```
1132     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1133         \bool_set_true:N \l_@@_width_used_bool ,
1134     width .value_required:n = true ,
1135     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1136     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1137     tabularnote .value_required:n = true ,
1138     caption .tl_set:N = \l_@@_caption_tl ,
1139     caption .value_required:n = true ,
1140     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1141     short-caption .value_required:n = true ,
1142     label .tl_set:N = \l_@@_label_tl ,
1143     label .value_required:n = true ,
1144     last-col .code:n = \tl_if_empty:nF { #1 }
1145         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1146         \int_zero:N \l_@@_last_col_int ,
1147     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1148     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1149     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1150 }
```

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
```

```

1151 \keys_define:nn { nicematrix / CodeAfter }
1152 {
1153   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1154   delimiters / color .value_required:n = true ,
1155   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1156   rules .value_required:n = true ,
1157   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1158   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1159   sub-matrix .value_required:n = true ,
1160   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1161 }

```

## 8 Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:-\@@_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}`).

```

1162 \cs_new_protected:Npn \@@_cell_begin:
1163 {

```

`\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.

```
1164   \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).

```
1165   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1166   \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.

```

1167   \int_compare:nNnT \c@jCol = \c_one_int
1168   { \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:`.

```
1169   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1170   \@@_tuning_not_tabular_begin:
1171   \@@_tuning_first_row:
1172   \@@_tuning_last_row:
1173   \g_@@_row_style_tl
1174 }

```

The following command will be nullified unless there is a first row. Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT \c@iRow
  {
    \int_compare:nNnT \c@jCol > 0
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

1175 \cs_new_protected:Npn \@@_tuning_first_row:
1176 {
1177   \if_int_compare:w \c@iRow = \c_zero_int
1178   \if_int_compare:w \c@jCol > \c_zero_int
1179   \l_@@_code_for_first_row_tl
1180   \xglobal \colorlet { nicematrix-first-row } { . }
1181   \fi:
1182   \fi:
1183 }

```

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

```

\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNnT \c@iRow = \l_@@_last_row_int
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}

```

We will use a version a little more efficient.

```

1184 \cs_new_protected:Npn \@@_tuning_last_row:
1185 {
1186   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1187   \l_@@_code_for_last_row_tl
1188   \xglobal \colorlet { nicematrix-last-row } { . }
1189   \fi:
1190 }

```

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

```

1191 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1192 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1193 {
1194   \m@th % added 2024/11/21
1195   \c_math_toggle_token

```

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

```

1196   \@@_tuning_key_small:
1197 }
1198 \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.

```

1199 \cs_new_protected:Npn \@@_begin_of_row:
1200 {
1201   \int_gincr:N \c@iRow

```

```

1202 \dim_gset:nn \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1203 \dim_gset:nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1204 \dim_gset:nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1205 \pgfpicture
1206 \pgfrememberpicturepositiononpagetrue
1207 \pgfcoordinate
1208 { \@@_env: - row - \int_use:N \c@iRow - base }
1209 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1210 \str_if_empty:NF \l_@@_name_str
1211 {
1212   \pgfnodelalias
1213   { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1214   { \@@_env: - row - \int_use:N \c@iRow - base }
1215 }
1216 \endpgfpicture
1217 }

```

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.

```

1218 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1219 {
1220   \int_if_zero:nTF \c@iRow
1221   {
1222     \dim_compare:nNnT { \box_dp:N \l_@@_cell_box } > \g_@@_dp_row_zero_dim
1223     { \dim_gset:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1224     \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1225     { \dim_gset:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1226   }
1227   {
1228     \int_compare:nNnT \c@iRow = \c_one_int
1229     {
1230       \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_one_dim
1231       { \dim_gset:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1232     }
1233   }
1234 }
1235 \cs_new_protected:Npn \@@_rotate_cell_box:
1236 {
1237   \box_rotate:nn \l_@@_cell_box { 90 }
1238   \bool_if:NTF \g_@@_rotate_c_bool
1239   {
1240     \hbox_set:nn \l_@@_cell_box
1241     {
1242       \m@th % add 2024/11/21
1243       \c_math_toggle_token
1244       \vcenter { \box_use:N \l_@@_cell_box }
1245       \c_math_toggle_token
1246     }
1247   }
1248   {
1249     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1250     {
1251       \vbox_set_top:nn \l_@@_cell_box
1252       {
1253         \vbox_to_zero:n { }
1254         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1255         \box_use:N \l_@@_cell_box
1256       }
1257     }
1258   }
1259 }

```

```

1258     }
1259     \bool_gset_false:N \g_@@_rotate_bool
1260     \bool_gset_false:N \g_@@_rotate_c_bool
1261 }
1262 \cs_new_protected:Npn \@@_adjust_size_box:
1263 {
1264     \dim_compare:nNtT \g_@@_blocks_wd_dim > \c_zero_dim
1265     {
1266         \box_set_wd:Nn \l_@@_cell_box
1267         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1268         \dim_gzero:N \g_@@_blocks_wd_dim
1269     }
1270     \dim_compare:nNtT \g_@@_blocks_dp_dim > \c_zero_dim
1271     {
1272         \box_set_dp:Nn \l_@@_cell_box
1273         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1274         \dim_gzero:N \g_@@_blocks_dp_dim
1275     }
1276     \dim_compare:nNtT \g_@@_blocks_ht_dim > \c_zero_dim
1277     {
1278         \box_set_ht:Nn \l_@@_cell_box
1279         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1280         \dim_gzero:N \g_@@_blocks_ht_dim
1281     }
1282 }
1283 \cs_new_protected:Npn \@@_cell_end:
1284 {

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1298     \@@_update_max_cell_width:

```

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

```

1299     \@@_update_for_first_and_last_row:

```

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

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

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

- for the columns of type p, m, b, V (of `varwidth`) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`

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

```

1300   \bool_if:NTF \g_@@_empty_cell_bool
1301   { \box_use_drop:N \l_@@_cell_box }
1302   {
1303     \bool_if:NTF \g_@@_not_empty_cell_bool
1304     \@@_print_node_cell:
1305     {
1306       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1307       \@@_print_node_cell:
1308       { \box_use_drop:N \l_@@_cell_box }
1309     }
1310   }
1311   \int_compare:nNnT \c_jCol > \g_@@_col_total_int
1312   { \int_gset_eq:NN \g_@@_col_total_int \c_jCol }
1313   \bool_gset_false:N \g_@@_empty_cell_bool
1314   \bool_gset_false:N \g_@@_not_empty_cell_bool
1315 }

```

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

```

1316 \cs_new_protected:Npn \@@_update_max_cell_width:
1317 {
1318   \dim_gset:Nn \g_@@_max_cell_width_dim
1319   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1320 }

```

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`).

```

1321 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1322 {
1323   \@@_math_toggle:
1324   \hbox_set_end:
1325   \bool_if:NF \g_@@_rotate_bool
1326   {
1327     \hbox_set:Nn \l_@@_cell_box
1328     {
1329       \makebox [ \l_@@_col_width_dim ] [ s ]
1330       { \hbox_unpack_drop:N \l_@@_cell_box }
1331     }
1332   }
1333   \@@_cell_end_i:
1334 }

```

```

1335 \pgfset
1336 {
1337   nicematrix / cell-node /.style =
1338   {
1339     inner~sep = \c_zero_dim ,
1340     minimum~width = \c_zero_dim
1341   }
1342 }

```

In the cells of a column of type `S` (of `siunitx`), we have to wrap the command `\@@_node_for_cell:` inside a command of `siunitx` to enforce the correct horizontal alignment. In the cells of the columns with other columns type, we don't have to do that job. That's why we create a socket with its default plug (`identity`) and a plug when we have to do the wrapping.

```

1343 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1344 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1345 {
1346   \use:c
1347   {
1348     __siunitx_table_align_
1349     \bool_if:NTF \l__siunitx_table_text_bool
1350     \l__siunitx_table_align_text_tl
1351     \l__siunitx_table_align_number_tl
1352     :n
1353   }
1354   { #1 }
1355 }
1356 \cs_new_protected:Npn \@@_print_node_cell:
1357 { \socket_use:nn { nicematrix / siunitx-wrap } { \@@_node_for_cell: } }

```

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

```

1358 \cs_new_protected:Npn \@@_node_for_cell:
1359 {
1360   \pgfpicture
1361   \pgfsetbaseline \c_zero_dim
1362   \pgfrememberpicturepositiononpagetrue
1363   \pgfset { nicematrix / cell-node }
1364   \pgfnode
1365   { rectangle }
1366   { base }
1367   {

```

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).

```

1368     \set@color
1369     \box_use_drop:N \l_@@_cell_box
1370   }
1371   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1372   { \l_@@_pgf_node_code_tl }
1373   \str_if_empty:NF \l_@@_name_str
1374   {
1375     \pgfnodealias
1376     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1377     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1378   }
1379   \endpgfpicture
1380 }

```

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.

```

1381 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1382 {
1383   \cs_new_protected:Npn \@@_patch_node_for_cell:
1384   {
1385     \hbox_set:Nn \l_@@_cell_box
1386     {
1387       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1388       \hbox_overlap_left:n
1389       {
1390         \pgfsys@markposition
1391         { \@@_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`, `dvips`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

1392         #1
1393     }
1394     \box_use:N \l_@@_cell_box
1395     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1396     \hbox_overlap_left:n
1397     {
1398         \pgfsys@markposition
1399         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1400         #1
1401     }
1402 }
1403 }
1404 }

```

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

```

1405 \bool_if:nTF { \sys_if_engine_xetex_p: || \cs_if_free_p:N \pdfoutput }
1406 {
1407     \@@_patch_node_for_cell:n
1408     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1409 }
1410 { \@@_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).

```

1411 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1412 {
1413     \bool_if:nTF { #1 } { \tl_gput_left:ce \tl_gput_right:ce
1414         { g_@@_ #2 _ lines _ tl }
1415         {
1416             \use:c { @@ _ draw _ #2 : nnn }
1417             { \int_use:N \c@iRow }
1418             { \int_use:N \c@jCol }
1419             { \exp_not:n { #3 } }
1420         }
1421     }

1422 \cs_generate_variant:Nn \@@_array:n { o }
1423 \cs_new_protected:Npn \@@_array:n
1424 {
1425     % \begin{macrocode}
1426     \dim_set:Nn \col@sep

```

```

1427 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1428 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1429 { \cs_set_nopar:Npn \@halignto { } }
1430 { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1431 \@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:eeTF is fully expandable and we need something fully expandable here. \str\_if\_eq:ee(TF) is faster than \str\_if\_eq:nn(TF).

```

1432 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1433 }

```

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.

```

1434 \bool_if:NTF
1435 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1436 { \cs_set_eq:NN \@_old_ar@ialign: \ar@ialign }
1437 { \cs_set_eq:NN \@_old_ialign: \ialign }

```

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

```

1438 \cs_new_protected:Npn \@_create_row_node:
1439 {
1440   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1441   {
1442     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1443     \@_create_row_node_i:
1444   }
1445 }
1446 \cs_new_protected:Npn \@_create_row_node_i:
1447 {

```

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

```

1448 \hbox
1449 {
1450   \bool_if:NT \l_@@_code_before_bool
1451   {
1452     \vtop
1453     {
1454       \skip_vertical:N 0.5\arrayrulewidth
1455       \pgfsys@markposition
1456       { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1457       \skip_vertical:N -0.5\arrayrulewidth
1458     }
1459   }
1460   \pgfpicture
1461   \pgfrememberpicturerepositiononpagetrue
1462   \pgfcoordinate { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1463   { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1464   \str_if_empty:NF \l_@@_name_str
1465   {
1466     \pgfnodealias
1467     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1468     { \@_env: - row - \int_eval:n { \c@iRow + 1 } }
1469   }
1470   \endpgfpicture
1471 }
1472 }

```

```

1473 \cs_new_protected:Npn \@@_in_everycr:
1474 {
1475   \bool_if:NT \c_@@_recent_array_bool
1476   {
1477     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1478     \tbl_update_cell_data_for_next_row:
1479   }
1480   \int_gzero:N \c@jCol
1481   \bool_gset_false:N \g_@@_after_col_zero_bool
1482   \bool_if:NF \g_@@_row_of_col_done_bool
1483   {
1484     \@@_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).

```

1485     \clist_if_empty:NF \l_@@_hlines_clist
1486     {
1487       \str_if_eq:eeF \l_@@_hlines_clist { all }
1488       {
1489         \clist_if_in:NeT
1490         \l_@@_hlines_clist
1491         { \int_eval:n { \c@iRow + 1 } }
1492       }
1493     }

```

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.

```

1494       \int_compare:nNnT \c@iRow > { -1 }
1495       {
1496         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1497         { \hrule height \arrayrulewidth width \c_zero_dim }
1498       }
1499     }
1500   }
1501 }
1502 }

```

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

```

1503 \cs_set_protected:Npn \@@_renew_dots:
1504 {
1505   \cs_set_eq:NN \ldots \@@_Ldots
1506   \cs_set_eq:NN \cdots \@@_Cdots
1507   \cs_set_eq:NN \vdots \@@_Vdots
1508   \cs_set_eq:NN \ddots \@@_Ddots
1509   \cs_set_eq:NN \iddots \@@_Iddots
1510   \cs_set_eq:NN \dots \@@_Ldots
1511   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1512 }

```

The following code has been simplified in the version 6.29a.

```

1513 \hook_gput_code:nnn { begindocument } { . }
1514 {
1515   \IfPackageLoadedTF { colortbl }
1516   {
1517     \cs_set_protected:Npn \@@_everycr:
1518     { \CT@everycr { \noalign { \@@_in_everycr: } } }
1519   }
1520   {
1521     \cs_new_protected:Npn \@@_everycr:
1522     { \everycr { \noalign { \@@_in_everycr: } } }
1523   }
1524 }

```

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>.

```

1525 \hook_gput_code:nnn { begindocument } { . }
1526 {
1527   \IfPackageLoadedTF { booktabs }
1528   {
1529     \cs_new_protected:Npn \@_patch_booktabs:
1530     { \tl_put_left:Nn \@BTnormal \@_create_row_node_i: }
1531   }
1532   { \cs_new_protected:Npn \@_patch_booktabs: { } }
1533 }

```

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`.

```

1534 \cs_new_protected:Npn \@_some_initialization:
1535 {
1536   \@_everycr:
1537   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1538   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1539   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1540   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1541   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1542   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1543 }

1544 \cs_new_protected:Npn \@_pre_array_ii:
1545 {

```

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

```

1546   \int_gzero:N \g_@@_total_X_weight_int
1547   \@_expand_clist:N \l_@@_hlines_clist
1548   \@_expand_clist:N \l_@@_vlines_clist
1549   \@_patch_booktabs:
1550   \box_clear_new:N \l_@@_cell_box
1551   \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}`).

```

1552   \bool_if:NT \l_@@_small_bool
1553   {
1554     \cs_set_nopar:Npn \arraystretch { 0.47 }
1555     \dim_set:Nn \arraycolsep { 1.45 pt }

```

---

<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}`.

By default, `\@@_tuning_key_small:` is no-op.

```

1556     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1557   }

1558   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1559   {
1560     \tl_put_right:Nn \@@_begin_of_row:
1561       {
1562         \pgfsys@markposition
1563         { \@@_env: - row - \int_use:N \c@iRow - base }
1564       }
1565   }

```

The environment `{array}` (since version 2.6) uses internally the command `\ar@ialign` (and previously, it was `\ialign`). We change that command for several reasons. In particular, `\ar@ialign` sets `\everycr` to `{ }` and we *need* to change the value of `\everycr`.

```

1566   \bool_if:nTF
1567   { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1568   {
1569     \cs_set_nopar:Npn \ar@ialign
1570     {
1571       \bool_if:NT \c_@@_testphase_table_bool
1572       \tbl_init_cell_data_for_table:
1573       \@@_some_initialization:
1574       \dim_zero:N \tabskip

```

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`.

```

1575       \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1576       \halign
1577     }
1578   }

```

The following part should be deleted when we will delete the boolean `\c_@@_recent_array_bool` (when we consider the version 2.6a of `array` is required). Moreover, `revtex4-2` modifies `array` and provides commands which are meant to be the standard version of `array` but, at the date of november 2024, these commands corresponds to the *old* version of `array`, that is to say without the `\ar@ialign`.

```

1579   {
1580     \cs_set_nopar:Npn \ialign
1581     {
1582       \@@_some_initialization:
1583       \dim_zero:N \tabskip
1584       \cs_set_eq:NN \ialign \@@_old_ialign:
1585       \halign
1586     }
1587   }

```

It seems that there is a problem when `nicematrix` is used with in `revtex4-2` with the package `colortbl` loaded. The following code prevent that problem but it does *not* treat the actual problem! It's only a patch *ad hoc*.

That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```

1588   \bool_if:NT \c_@@_revtex_bool
1589   {
1590     \IfPackageLoadedT { colortbl }
1591     { \cs_set_protected:Npn \CT@setup { } }
1592   }

```

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).

```

1593   \cs_set_eq:NN \@@_old_ldots \ldots
1594   \cs_set_eq:NN \@@_old_cdots \cdots

```

```

1595 \cs_set_eq:NN \@@_old_vdots \vdots
1596 \cs_set_eq:NN \@@_old_ddots \ddots
1597 \cs_set_eq:NN \@@_old_iddots \iddots
1598 \bool_if:NTF \l_@@_standard_cline_bool
1599 { \cs_set_eq:NN \cline \@@_standard_cline }
1600 { \cs_set_eq:NN \cline \@@_cline }
1601 \cs_set_eq:NN \Ldots \@@_Ldots
1602 \cs_set_eq:NN \Cdots \@@_Cdots
1603 \cs_set_eq:NN \Vdots \@@_Vdots
1604 \cs_set_eq:NN \Ddots \@@_Ddots
1605 \cs_set_eq:NN \Iddots \@@_Iddots
1606 \cs_set_eq:NN \Hline \@@_Hline:
1607 \cs_set_eq:NN \Hspace \@@_Hspace:
1608 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1609 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1610 \cs_set_eq:NN \Block \@@_Block:
1611 \cs_set_eq:NN \rotate \@@_rotate:
1612 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1613 \cs_set_eq:NN \dotfill \@@_dotfill:
1614 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1615 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1616 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1617 \cs_set_eq:NN \TopRule \@@_TopRule
1618 \cs_set_eq:NN \MidRule \@@_MidRule
1619 \cs_set_eq:NN \BottomRule \@@_BottomRule
1620 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1621 \cs_set_eq:NN \Hbrace \@@_Hbrace
1622 \cs_set_eq:NN \Vbrace \@@_Vbrace
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  $n$ ) correspondant 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:` 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:` 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_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1684     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1685     \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1686     { \dim_gset: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_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1693 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1694 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1695 \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.

```

1696 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value  $-2$  is important.

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

```

1697 \@@_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.

```

1698 \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]`).

```

1699 \dim_zero_new:N \l_@@_left_delim_dim
1700 \dim_zero_new:N \l_@@_right_delim_dim
1701 \bool_if:NTF \g_@@_delims_bool
1702 {

```

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

```

1703   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1704   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1705   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1706   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1707 }
1708 {
1709   \dim_gset:Nn \l_@@_left_delim_dim
1710   { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1711   \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1712 }

```



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).

```

1713 \hbox_set:Nw \l_@@_the_array_box
1714 \skip_horizontal:N \l_@@_left_margin_dim
1715 \skip_horizontal:N \l_@@_extra_left_margin_dim
1716 \bool_if:NT \c_@@_recent_array_bool
1717 { \UseTaggingSocket { tbl / hmode / begin } }

```

The following code is a workaround to specify to the tagging system that the following code is *fake math* (it raises `\l__math_fakemath_bool` in recent versions of LaTeX).

```

1718 \m@th
1719 \c_math_toggle_token
1720 \bool_if:NTF \l_@@_light_syntax_bool
1721 { \use:c { @@-light-syntax } }
1722 { \use:c { @@-normal-syntax } }
1723 }

```

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

```

1724 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1725 {
1726   \tl_set:Nn \l_tmpa_tl { #1 }
1727   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1728   { \@@_rescan_for_spanish:N \l_tmpa_tl }
1729   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1730   \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...

```

1731 \@@_pre_array:
1732 }

```

## 9 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 alone only for legibility).

```

1733 \cs_new_protected:Npn \@@_pre_code_before:
1734 {

```

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).

```

1735 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1736 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1737 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1738 \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 1247 of `pgfmanual.pdf`, version 3.1.10.

```

1739 \pgfsys@markposition { \@@_env: - position }
1740 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1741 \pgfpicture
1742 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1743 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1744 {
1745   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1746   \pgfcoordinate { \@@_env: - row - ##1 }
1747   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1748 }

```

Now, the recreation of the col nodes.

```

1749 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1750 {
1751   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1752   \pgfcoordinate { \@@_env: - col - ##1 }
1753   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1754 }

```

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

```

1755 \@@_create_diag_nodes:

```

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

```

1756 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1757 \endpgfpicture

```

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

```

1758 \@@_create_blocks_nodes:
1759 \IfPackageLoadedT { tikz }
1760 {
1761   \tikzset
1762   {
1763     every-picture / .style =
1764     { overlay , name-prefix = \@@_env: - }
1765   }
1766 }
1767 \cs_set_eq:NN \cellcolor \@@_cellcolor
1768 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1769 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1770 \cs_set_eq:NN \rowcolor \@@_rowcolor
1771 \cs_set_eq:NN \rowcolors \@@_rowcolors
1772 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1773 \cs_set_eq:NN \arraycolor \@@_arraycolor
1774 \cs_set_eq:NN \columncolor \@@_columncolor
1775 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1776 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1777 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1778 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1779 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1780 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1781 }

```

```

1782 \cs_new_protected:Npn \@@_exec_code_before:
1783 {

```

We mark the cells which are in the (empty) corners because those cells must not be colored. We should try to find a way to detected whether we actually have coloring instructions to execute...

```

1784 \clist_map_inline:Nn \l_@@_corners_cells_clist
1785 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1786 \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`.

```
1787 \@@_add_to_colors_seq:nn { { nocolor } } { }
1788 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1789 \group_begin:
```

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

```
1790 \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`).

```
1791 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1792 { \@@_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`.

```
1793 \exp_last_unbraced:No \@@_CodeBefore_keys:
1794 \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.

```
1795 \@@_actually_color:
1796 \l_@@_code_before_tl
1797 \q_stop
1798 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1799 \group_end:
1800 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1801 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1802 }
```

```
1803 \keys_define:nn { nicematrix / CodeBefore }
1804 {
1805   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1806   create-cell-nodes .default:n = true ,
1807   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1808   sub-matrix .value_required:n = true ,
1809   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1810   delimiters / color .value_required:n = true ,
1811   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1812 }
1813 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1814 {
1815   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1816   \@@_CodeBefore:w
1817 }
```

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.

```
1818 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1819 {
1820   \bool_if:NT \g_@@_aux_found_bool
1821   {
1822     \@@_pre_code_before:
1823     #1
```

```

1824     }
1825 }

```

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.

```

1826 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1827 {
1828   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1829   {
1830     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1831     \pgfcoordinate { \@@_env: - row - ##1 - base }
1832     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1833     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1834     {
1835       \cs_if_exist:cT
1836       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1837       {
1838         \pgfsys@getposition
1839         { \@@_env: - ##1 - ####1 - NW }
1840         \@@_node_position:
1841         \pgfsys@getposition
1842         { \@@_env: - ##1 - ####1 - SE }
1843         \@@_node_position_i:
1844         \@@_pgf_rect_node:nnn
1845         { \@@_env: - ##1 - ####1 }
1846         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1847         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1848       }
1849     }
1850   }
1851   \int_step_inline:nn \c@iRow
1852   {
1853     \pgfnodealias
1854     { \@@_env: - ##1 - last }
1855     { \@@_env: - ##1 - \int_use:N \c@jCol }
1856   }
1857   \int_step_inline:nn \c@jCol
1858   {
1859     \pgfnodealias
1860     { \@@_env: - last - ##1 }
1861     { \@@_env: - \int_use:N \c@iRow - ##1 }
1862   }
1863   \@@_create_extra_nodes:
1864 }

1865 \cs_new_protected:Npn \@@_create_blocks_nodes:
1866 {
1867   \pgfpicture
1868   \pgf@relevantforpicturesizefalse
1869   \pgfrememberpicturepositiononpagetrue
1870   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1871   { \@@_create_one_block_node:nnnnn ##1 }
1872   \endpgfpicture
1873 }

```

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>

---

<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).

```

1874 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1875 {
1876   \tl_if_empty:nF { #5 }
1877   {
1878     \@@_qpoint:n { col - #2 }
1879     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1880     \@@_qpoint:n { #1 }
1881     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1882     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1883     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1884     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1885     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1886     \@@_pgf_rect_node:nnnnn
1887     { \@@_env: - #5 }
1888     { \dim_use:N \l_tmpa_dim }
1889     { \dim_use:N \l_tmpb_dim }
1890     { \dim_use:N \l_@@_tmpc_dim }
1891     { \dim_use:N \l_@@_tmpd_dim }
1892   }
1893 }

1894 \cs_new_protected:Npn \@@_patch_for_revtext:
1895 {
1896   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1897   \cs_set_eq:NN \@array \@array@array
1898   \cs_set_eq:NN \@tabular \@tabular@array
1899   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1900   \cs_set_eq:NN \array \array@array
1901   \cs_set_eq:NN \endarray \endarray@array
1902   \cs_set:Npn \endtabular { \endarray $\egroup } % $
1903   \cs_set_eq:NN \@mkpream \@mkpream@array
1904   \cs_set_eq:NN \@classx \@classx@array
1905   \cs_set_eq:NN \insert@column \insert@column@array
1906   \cs_set_eq:NN \@arraycr \@arraycr@array
1907   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1908   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1909 }

```

## 10 The environment {NiceArrayWithDelims}

```

1910 \NewDocumentEnvironment { NiceArrayWithDelims }
1911 { m m O { } m ! O { } t \CodeBefore }
1912 {
1913   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1914   \@@_provide_pgfsyspdfmark:
1915   \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.

```

1916   \bgroup

1917   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1918   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1919   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1920   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1921   \int_gzero:N \g_@@_block_box_int
1922   \dim_zero:N \g_@@_width_last_col_dim

```

```

1923 \dim_zero:N \g_@@_width_first_col_dim
1924 \bool_gset_false:N \g_@@_row_of_col_done_bool
1925 \str_if_empty:NT \g_@@_name_env_str
1926 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1927 \bool_if:NTF \l_@@_tabular_bool
1928 \mode_leave_vertical:
1929 \@@_test_if_math_mode:
1930 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1931 \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.

```

1932 \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.

```

1933 \cs_if_exist:NT \tikz@library@external@loaded
1934 {
1935 \tikzexternaldisable
1936 \cs_if_exist:NT \ifstandalone
1937 { \tikzset { external / optimize = false } }
1938 }

```

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

```

1939 \int_gincr:N \g_@@_env_int
1940 \bool_if:NF \l_@@_block_auto_columns_width_bool
1941 { \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.

```

1942 \seq_gclear:N \g_@@_blocks_seq
1943 \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`.

```

1944 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1945 \seq_gclear:N \g_@@_pos_of_xdots_seq
1946 \tl_gclear_new:N \g_@@_code_before_tl
1947 \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.

```

1948 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1949 {
1950 \bool_gset_true:N \g_@@_aux_found_bool
1951 \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1952 }
1953 { \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.

```

1954 \tl_gclear:N \g_@@_aux_tl
1955 \tl_if_empty:NF \g_@@_code_before_tl
1956 {
1957 \bool_set_true:N \l_@@_code_before_bool
1958 \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1959 }
1960 \tl_if_empty:NF \g_@@_pre_code_before_tl

```

---

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

```
1961 { \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.

```
1962 \bool_if:NTF \g_@@_delims_bool
1963 { \keys_set:nn { nicematrix / pNiceArray } }
1964 { \keys_set:nn { nicematrix / NiceArray } }
1965 { #3 , #5 }
```

```
1966 \@@_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:.

```
1967 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1968 }
```

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

```
1969 {
1970   \bool_if:NTF \l_@@_light_syntax_bool
1971   { \use:c { end @@-light-syntax } }
1972   { \use:c { end @@-normal-syntax } }
1973   \c_math_toggle_token
1974   \skip_horizontal:N \l_@@_right_margin_dim
1975   \skip_horizontal:N \l_@@_extra_right_margin_dim
1976
1977   % awful workaround
1978   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1979   {
1980     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1981     {
1982       \skip_horizontal:N - \l_@@_columns_width_dim
1983       \bool_if:NTF \l_@@_tabular_bool
1984       { \skip_horizontal:n { - 2 \tabcolsep } }
1985       { \skip_horizontal:n { - 2 \arraycolsep } }
1986     }
1987   }
1988   \hbox_set_end:
1989   \bool_if:NT \c_@@_recent_array_bool
1990   { \UseTaggingSocket { tbl / hmode / 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.

```
1991 \bool_if:NT \l_@@_width_used_bool
1992 {
1993   \int_if_zero:nT \g_@@_total_X_weight_int
1994   { \@@_error_or_warning:n { width-without-X-columns } }
1995 }
```

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.

```
1996 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1997 { \@@_compute_width_X: }
```

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).

```
1998 \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```

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

```

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>

```

2009   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2010   \bool_if:NTF \g_@@_last_col_found_bool
2011   { \int_gdecr:N \c@jCol }
2012   {
2013     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2014     { \@@_error:n { last-col-not-used } }
2015   }

```

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

```

2016   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2017   \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. 90).

```

2018   \int_if_zero:nT \l_@@_first_col_int
2019   { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2020   \bool_if:nTF { ! \g_@@_delims_bool }
2021   {
2022     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2023     \@@_use_arraybox_with_notes_c:
2024     {
2025       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2026       \@@_use_arraybox_with_notes_b:
2027       \@@_use_arraybox_with_notes:
2028     }
2029   }

```

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).

```

2030   {
2031     \int_if_zero:nTF \l_@@_first_row_int
2032     {
2033       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2034       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2035     }
2036     { \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>

```

2037   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2038   {
2039     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2040     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2041   }

```

<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).



```

2042         { \dim_zero:N \l_tmpb_dim }
2043     \hbox_set:Nn \l_tmpa_box
2044     {
2045         \m@th % added 2024/11/21
2046         \c_math_toggle_token
2047         \@@_color:o \l_@@_delimiters_color_tl
2048         \exp_after:wN \left \g_@@_left_delim_tl
2049         \vcenter
2050         {

```

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

```

2051         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2052         \hbox
2053         {
2054             \bool_if:NTF \l_@@_tabular_bool
2055             { \skip_horizontal:N -\tabcolsep }
2056             { \skip_horizontal:N -\arraycolsep }
2057             \@@_use_arraybox_with_notes_c:
2058             \bool_if:NTF \l_@@_tabular_bool
2059             { \skip_horizontal:N -\tabcolsep }
2060             { \skip_horizontal:N -\arraycolsep }
2061         }

```

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

```

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

```

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

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

```

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

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 91).

```

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

```

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

```

2084     \egroup

```

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

```

2085     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2086     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2087     \iow_now:Ne \@mainaux
2088     {
2089         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }

```

```

2090      { \exp_not:o \g_@@_aux_tl }
2091    }
2092    \iow_now:Nn \@mainaux { \ExplSyntaxOff }

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

```

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

The following command will be used only once. We have written that command for legibility. 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  $n$ .

```

2095 \cs_new_protected:Npn \@_compute_width_X:
2096 {
2097   \tl_gput_right:Ne \g_@@_aux_tl
2098   {
2099     \bool_set_true:N \l_@@_X_columns_aux_bool
2100     \dim_set:Nn \l_@@_X_columns_dim
2101     {
2102       \dim_compare:nNnTF
2103       {
2104         \dim_abs:n
2105         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2106       }
2107       <
2108       { 0.001 pt }
2109       { \dim_use:N \l_@@_X_columns_dim }
2110       {
2111         \dim_eval:n
2112         {
2113           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2114           / \int_use:N \g_@@_total_X_weight_int
2115           + \l_@@_X_columns_dim
2116         }
2117       }
2118     }
2119   }
2120 }

```

## 11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which 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.

```

2121 \cs_new_protected:Npn \@_transform_preamble:
2122 {
2123   \@_transform_preamble_i:
2124   \@_transform_preamble_ii:
2125 }

2126 \cs_new_protected:Npn \@_transform_preamble_i:
2127 {
2128   \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`).

```

2129   \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.

```
2130 \bool_gset_false:N \g_tmpb_bool
```

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

```
2131 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2132 \int_zero:N \l_tmpa_int
2133 \tl_gclear:N \g_@@_array_preamble_tl
2134 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2135 {
2136   \tl_gset:Nn \g_@@_array_preamble_tl
2137     { ! { \skip_horizontal:N \arrayrulewidth } }
2138 }
2139 {
2140   \clist_if_in:NnT \l_@@_vlines_clist 1
2141   {
2142     \tl_gset:Nn \g_@@_array_preamble_tl
2143       { ! { \skip_horizontal:N \arrayrulewidth } }
2144   }
2145 }
```

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

```
2146 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2147 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2148 \@@_replace_columncolor:
2149 }
```

```
2150 \hook_gput_code:nnn { begindocument } { . }
2151 {
2152   \IfPackageLoadedTF { colortbl }
2153   {
```

When `colortbl` is used, we have to catch the tokens `\columncolor` in the preamble because, otherwise, `colortbl` will catch them and the colored panels won't be drawn by `nicematrix` but by `colortbl` (with an output which is not perfect).

```
2154   \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2155   \cs_new_protected:Npn \@@_replace_columncolor:
2156   {
2157     \regex_replace_all:NnN
2158       \c_@@_columncolor_regex
2159       { \c { @@_columncolor_preamble } }
2160       \g_@@_array_preamble_tl
2161   }
2162 }
2163 {
2164   \cs_new_protected:Npn \@@_replace_columncolor:
2165   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2166 }
2167 }
```

```
2168 \cs_new_protected:Npn \@@_transform_preamble_ii:
2169 {
```

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

```
2170 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2171 {
2172   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
```

```

2173         { \bool_gset_true:N \g_@@_delims_bool }
2174     }
2175     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2176     \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).

```

2177     \int_if_zero:nTF \l_@@_first_col_int
2178     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2179     {
2180         \bool_if:NF \g_@@_delims_bool
2181         {
2182             \bool_if:NF \l_@@_tabular_bool
2183             {
2184                 \clist_if_empty:NT \l_@@_vlines_clist
2185                 {
2186                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2187                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2188                 }
2189             }
2190         }
2191     }
2192     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2193     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2194     {
2195         \bool_if:NF \g_@@_delims_bool
2196         {
2197             \bool_if:NF \l_@@_tabular_bool
2198             {
2199                 \clist_if_empty:NT \l_@@_vlines_clist
2200                 {
2201                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2202                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2203                 }
2204             }
2205         }
2206     }

```

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).

```

2207     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2208     {

```

If the tagging of the tabulars is done (part of the Tagging Project), you don’t activate that mechanism because it would create a dummy column of tagged empty cells.

```

2209         \bool_if:NF \c_@@_testphase_table_bool
2210         {
2211             \tl_gput_right:Nn \g_@@_array_preamble_tl
2212             { > { \@@_error_too_much_cols: } l }
2213         }
2214     }
2215 }

```

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.

```

2216 \cs_new_protected:Npn \@@_rec_preamble:n #1
2217 {

```

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>

```
2218 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2219 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2220 {
```

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

```
2221 \cs_if_exist:cTF { NC @ find @ #1 }
2222 {
2223 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2224 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2225 }
2226 {
2227 \str_if_eq:nnTF { #1 } { S }
2228 { \@@_fatal:n { unknown~column~type~S } }
2229 { \@@_fatal:nn { unknown~column~type } { #1 } }
2230 }
2231 }
2232 }
```

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

```
2233 \cs_new_protected:Npn \@@_c #1
2234 {
2235 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2236 \tl_gclear:N \g_@@_pre_cell_tl
2237 \tl_gput_right:Nn \g_@@_array_preamble_tl
2238 { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2239 \int_gincr:N \c@jCol
2240 \@@_rec_preamble_after_col:n
2241 }
2242 \cs_new_protected:Npn \@@_l #1
2243 {
2244 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2245 \tl_gclear:N \g_@@_pre_cell_tl
2246 \tl_gput_right:Nn \g_@@_array_preamble_tl
2247 {
2248 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2249 l
2250 < \@@_cell_end:
2251 }
2252 \int_gincr:N \c@jCol
2253 \@@_rec_preamble_after_col:n
2254 }
2255 \cs_new_protected:Npn \@@_r #1
2256 {
2257 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2258 \tl_gclear:N \g_@@_pre_cell_tl
2259 \tl_gput_right:Nn \g_@@_array_preamble_tl
2260 {
2261 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2262 r
2263 < \@@_cell_end:
2264 }
2265 \int_gincr:N \c@jCol
2266 \@@_rec_preamble_after_col:n
2267 }
```

---

<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`.

For ! and @

```

2268 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2269 {
2270   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2271   \@@_rec_preamble:n
2272 }
2273 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2274 \cs_new_protected:cpn { @@ _ | } #1
2275 {

```

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

```

2276   \int_incr:N \l_tmpa_int
2277   \@@_make_preamble_i_i:n
2278 }

2279 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2280 {
2281   \str_if_eq:nnTF { #1 } { | }
2282   { \use:c { @@ _ | } | }
2283   { \@@_make_preamble_i_ii:nn { } #1 }
2284 }

2285 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2286 {
2287   \str_if_eq:nnTF { #2 } { [ ] }
2288   { \@@_make_preamble_i_iii:nn { #1 } [ ] }
2289   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2290 }

2291 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2292 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2293 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2294 {
2295   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2296   \tl_gput_right:Ne \g_@@_array_preamble_tl
2297   {

```

Here, the command \dim\_use:N is mandatory.

```

2298     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2299   }
2300   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2301   {
2302     \@@_vline:n
2303     {
2304       position = \int_eval:n { \c@jCol + 1 } ,
2305       multiplicity = \int_use:N \l_tmpa_int ,
2306       total-width = \dim_use:N \l_@@_rule_width_dim ,
2307       #2
2308     }

```

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.

```

2309   }
2310   \int_zero:N \l_tmpa_int
2311   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2312   \@@_rec_preamble:n #1
2313 }

2314 \cs_new_protected:cpn { @@ _ > } #1 #2
2315 {
2316   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2317   \@@_rec_preamble:n
2318 }

```

```
2319 \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.

```
2320 \keys_define:nn { nicematrix / p-column }
2321 {
2322   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2323   r .value_forbidden:n = true ,
2324   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2325   c .value_forbidden:n = true ,
2326   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2327   l .value_forbidden:n = true ,
2328   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2329   S .value_forbidden:n = true ,
2330   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2331   p .value_forbidden:n = true ,
2332   t .meta:n = p ,
2333   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2334   m .value_forbidden:n = true ,
2335   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2336   b .value_forbidden:n = true
2337 }
```

For `p` but also `b` and `m`.

```
2338 \cs_new_protected:Npn \@@_p #1
2339 {
2340   \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2341   \@@_make_preamble_ii_i:n
2342 }
2343 \cs_set_eq:NN \@@_b \@@_p
2344 \cs_set_eq:NN \@@_m \@@_p
2345 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2346 {
2347   \str_if_eq:nnTF { #1 } { [ ]
2348     { \@@_make_preamble_ii_ii:w [ ]
2349       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2350     }
2351   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2352     { \@@_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.

```
2353 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2354 {
```

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).

```
2355   \str_set:Nn \l_@@_hpos_col_str { j }
2356   \@@_keys_p_column:n { #1 }
2357   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2358 }
2359 \cs_new_protected:Npn \@@_keys_p_column:n #1
2360 { \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.

```
2361 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2362 {
2363   \use:e
```

```

2364 {
2365   \@@_make_preamble_ii_v:nnnnnnnn
2366   { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2367   { \dim_eval:n { #1 } }
2368   {

```

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.

```

2369   \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2370   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2371   {

```

Here, we use `\cs_set_nopar:Npn` instead of `\tl_set:Nn` for efficiency only.

```

2372   \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2373   { \str_lowercase:o \l_@@_hpos_col_str }
2374   }
2375   \IfPackageLoadedTF { ragged2e }
2376   {
2377     \str_case:on \l_@@_hpos_col_str
2378     {
2379       c { \exp_not:N \Centering }
2380       l { \exp_not:N \RaggedRight }
2381       r { \exp_not:N \RaggedLeft }
2382     }
2383   }
2384   {
2385     \str_case:on \l_@@_hpos_col_str
2386     {
2387       c { \exp_not:N \centering }
2388       l { \exp_not:N \raggedright }
2389       r { \exp_not:N \raggedleft }
2390     }
2391   }
2392   #3
2393   }
2394   { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2395   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2396   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2397   { #2 }
2398   {
2399     \str_case:onF \l_@@_hpos_col_str
2400     {
2401       { j } { c }
2402       { si } { c }
2403     }

```

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

```

2404     { \str_lowercase:o \l_@@_hpos_col_str }
2405   }
2406   }

```

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

```

2407   \int_gincr:N \c@jCol
2408   \@@_rec_preamble_after_col:n
2409   }

```

**#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*.

```

2410 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2411 {
2412   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2413   {
2414     \tl_gput_right:Nn \g_@@_array_preamble_tl
2415     { > \@@_test_if_empty_for_S: }
2416   }
2417   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2418   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2419   \tl_gclear:N \g_@@_pre_cell_tl
2420   \tl_gput_right:Nn \g_@@_array_preamble_tl
2421   {
2422     > {

```

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.

```

2423       \dim_set:Nn \l_@@_col_width_dim { #2 }
2424       \bool_if:NT \c_@@_testphase_table_bool
2425       { \tag_struct_begin:n { tag = Div } }
2426       \@@_cell_begin:

```

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

```

2427       \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

2428       \everypar
2429       {
2430         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2431         \everypar { }
2432       }
2433       \bool_if:NT \c_@@_testphase_table_bool \tagpdfpara0n

```

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

```

2434       #3

```

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

```

2435       \g_@@_row_style_tl
2436       \arraybackslash
2437       #5
2438     }
2439     #8
2440     < {
2441       #6

```

The following line has been taken from array.sty.

```

2442       \@finalstrut \@arstrutbox
2443       \use:c { end #7 }

```

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

```

2444       #4
2445       \@@_cell_end:
2446       \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2447     }
2448   }
2449 }

```

The cell always begins with `\ignorespaces` with `array` and that's why we retrieve that token.

```
2450 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2451 {
```

We open a special group with `\group_align_safe_begin:`. Thus, when `\peek_meaning:NTF` will read the `&` (when the cell is empty), that lecture won't trigger the end of the cell (since we are in a lower group...). If the end of cell was triggered, we would have other tokens in the TeX flow (and not `&`).

```
2452 \group_align_safe_begin:
2453 \peek_meaning:NTF &
2454 \@@_the_cell_is_empty:
2455 {
2456   \peek_meaning:NTF \\\
2457   \@@_the_cell_is_empty:
2458   {
2459     \peek_meaning:NTF \crcr
2460     \@@_the_cell_is_empty:
2461     \group_align_safe_end:
2462   }
2463 }
2464 }

2465 \cs_new_protected:Npn \@@_the_cell_is_empty:
2466 {
2467   \group_align_safe_end:
2468   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2469   {
```

Be careful: here, we can't merely use `\bool_gset_true: \g_@@_empty_cell_bool`, in particular because of the columns of type X.

```
2470   \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2471   \skip_horizontal:N \l_@@_col_width_dim
2472 }
2473 }

2474 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2475 {
2476   \peek_meaning:NT \__siunitx_table_skip:n
2477   { \bool_gset_true:N \g_@@_empty_cell_bool }
2478 }
```

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.

```
2479 \cs_new_protected:Npn \@@_center_cell_box:
2480 {
```

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

```
2481   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2482   {
2483     \int_compare:nNnT
2484     { \box_ht:N \l_@@_cell_box }
2485     >
```

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).

```
2486     { \box_ht:N \strutbox }
2487     {
2488       \hbox_set:Nn \l_@@_cell_box
2489       {
```

```

2490         \box_move_down:nn
2491         {
2492             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2493               + \baselineskip ) / 2
2494         }
2495         { \box_use:N \l_@@_cell_box }
2496     }
2497 }
2498 }
2499 }

```

For V (similar to the V of varwidth).

```

2500 \cs_new_protected:Npn \@@_V #1 #2
2501 {
2502     \str_if_eq:nnTF { #1 } { [ ] }
2503     { \@@_make_preamble_V_i:w [ ] }
2504     { \@@_make_preamble_V_i:w [ ] { #2 } }
2505 }
2506 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2507 { \@@_make_preamble_V_ii:nn { #1 } }
2508 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2509 {
2510     \str_set:Nn \l_@@_vpos_col_str { p }
2511     \str_set:Nn \l_@@_hpos_col_str { j }
2512     \@@_keys_p_column:n { #1 }
2513     \IfPackageLoadedTF { varwidth }
2514     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2515     {
2516         \@@_error_or_warning:n { varwidth~not~loaded }
2517         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2518     }
2519 }

```

For w and W

```

2520 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2521 \cs_new_protected: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.

```

2522 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2523 {
2524     \str_if_eq:nnTF { #3 } { s }
2525     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2526     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2527 }

```

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.

```

2528 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2529 {
2530     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2531     \tl_gclear:N \g_@@_pre_cell_tl
2532     \tl_gput_right:Nn \g_@@_array_preamble_tl
2533     {
2534         > {
2535             \dim_set:Nn \l_@@_col_width_dim { #2 }
2536             \@@_cell_begin:
2537             \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2538         }

```

```

2539         c
2540     < {
2541         \@@_cell_end_for_w_s:
2542         #1
2543         \@@_adjust_size_box:
2544         \box_use_drop:N \l_@@_cell_box
2545     }
2546 }
2547 \int_gincr:N \c@jCol
2548 \@@_rec_preamble_after_col:n
2549 }

```

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

```

2550 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2551 {
2552     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2553     \tl_gclear:N \g_@@_pre_cell_tl
2554     \tl_gput_right:Nn \g_@@_array_preamble_tl
2555     {
2556         > {

```

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.

```

2557         \dim_set:Nn \l_@@_col_width_dim { #4 }
2558         \hbox_set:Nw \l_@@_cell_box
2559         \@@_cell_begin:
2560         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2561     }
2562     c
2563     < {
2564         \@@_cell_end:
2565         \hbox_set_end:
2566         #1
2567         \@@_adjust_size_box:
2568         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2569     }
2570 }

```

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

```

2571     \int_gincr:N \c@jCol
2572     \@@_rec_preamble_after_col:n
2573 }

```

```

2574 \cs_new_protected:Npn \@@_special_W:
2575 {
2576     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2577     { \@@_warning:n { W-warning } }
2578 }

```

For S (of siunitx).

```

2579 \cs_new_protected:Npn \@@_S #1 #2
2580 {
2581     \str_if_eq:nnTF { #2 } { [ ] }
2582     { \@@_make_preamble_S:w [ ] }
2583     { \@@_make_preamble_S:w [ ] { #2 } }
2584 }
2585 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2586 { \@@_make_preamble_S:i:n { #1 } }
2587 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2588 {
2589     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2590     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl

```

```

2591 \tl_gclear:N \g_@@_pre_cell_tl
2592 \tl_gput_right:Nn \g_@@_array_preamble_tl
2593 {
2594 > {

```

In the cells of a column of type S, we have to wrap the command `\@@_node_for_cell:` for the horizontal alignment of the content of the cell (siunitx has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignment once again).

```

2595 \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2596 \keys_set:nn { siunitx } { #1 }
2597 \@@_cell_begin:
2598 \siunitx_cell_begin:w
2599 }
2600 c
2601 <
2602 {
2603 \siunitx_cell_end:

```

We want the value of `\l__siunitx_table_text_bool` available *after* `\@@_cell_end:` because we need it to know how to align our box after the construction of the PGF/TikZ node. That's why we use `\g_@@_cell_after_hook_tl` to reset the correct value of `\l__siunitx_table_text_bool` (of course, if will stay local within the cell of the underlying `\halign`).

```

2604 \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2605 {
2606 \bool_if:NTF \l__siunitx_table_text_bool
2607 \bool_set_true:N
2608 \bool_set_false:N
2609 \l__siunitx_table_text_bool
2610 }
2611 \@@_cell_end:
2612 }
2613 }

```

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

```

2614 \int_gincr:N \c@jCol
2615 \@@_rec_preamble_after_col:n
2616 }

```

For `(`, `[` and `\{`.

```

2617 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2618 {
2619 \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.

```

2620 \int_if_zero:nTF \c@jCol
2621 {
2622 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2623 {

```

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

```

2624 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2625 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2626 \@@_rec_preamble:n #2
2627 }
2628 {
2629 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2630 \@@_make_preamble_iv:nn { #1 } { #2 }
2631 }
2632 }
2633 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2634 }
2635 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2636 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }

```

```

2637 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2638 {
2639   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2640   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2641   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2642   {
2643     \@@_error:nn { delimiter~after~opening } { #2 }
2644     \@@_rec_preamble:n
2645   }
2646   { \@@_rec_preamble:n #2 }
2647 }

```

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

```

2648 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2649 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2650 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2651 {
2652   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2653   \tl_if_in:nnTF { ) ] \} } { #2 }
2654   { \@@_make_preamble_v:nnn #1 #2 }
2655   {
2656     \str_if_eq:nnTF { \@@_stop: } { #2 }
2657     {
2658       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2659       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2660       {
2661         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2662         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2663         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2664         \@@_rec_preamble:n #2
2665       }
2666     }
2667     {
2668       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2669       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2670       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2671       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2672       \@@_rec_preamble:n #2
2673     }
2674   }
2675 }
2676 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2677 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2678 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2679 {
2680   \str_if_eq:nnTF { \@@_stop: } { #3 }
2681   {
2682     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2683     {
2684       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2685       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2686       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2687       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2688     }
2689     {
2690       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2691       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2692       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }

```

```

2693         \@@_error:nn { double~closing~delimiter } { #2 }
2694     }
2695 }
2696 {
2697     \tl_gput_right:Nn \g_@@_pre_code_after_tl
2698     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2699     \@@_error:nn { double~closing~delimiter } { #2 }
2700     \@@_rec_preamble:n #3
2701 }
2702 }

2703 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2704 { \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 @{...}.

```

2705 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2706 {
2707     \str_if_eq:nnTF { #1 } { < }
2708     \@@_rec_preamble_after_col_i:n
2709     {
2710         \str_if_eq:nnTF { #1 } { @ }
2711         \@@_rec_preamble_after_col_ii:n
2712         {
2713             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2714             {
2715                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2716                 { ! { \skip_horizontal:N \arrayrulewidth } }
2717             }
2718             {
2719                 \clist_if_in:NnT \l_@@_vlines_clist
2720                 { \int_eval:n { \c@jCol + 1 } }
2721                 {
2722                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2723                     { ! { \skip_horizontal:N \arrayrulewidth } }
2724                 }
2725             }
2726             \@@_rec_preamble:n { #1 }
2727         }
2728     }
2729 }

2730 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2731 {
2732     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2733     \@@_rec_preamble_after_col:n
2734 }

```

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.

```

2735 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2736 {
2737     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2738     {
2739         \tl_gput_right:Nn \g_@@_array_preamble_tl
2740         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2741     }
2742     {
2743         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2744         {
2745             \tl_gput_right:Nn \g_@@_array_preamble_tl
2746             { @ { #1 \skip_horizontal:N \arrayrulewidth } }

```

```

2747     }
2748     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2749   }
2750   \@@_rec_preamble:n
2751 }

2752 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2753 {
2754   \tl_clear:N \l_tmpa_tl
2755   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2756   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2757 }

```

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

```

2758 \cs_new_protected: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.

```

2759 \cs_new_protected:Npn \@@_X #1 #2
2760 {
2761   \str_if_eq:nnTF { #2 } { [ ]
2762     { \@@_make_preamble_X:w [ ] }
2763     { \@@_make_preamble_X:w [ ] #2 }
2764   }
2765   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2766   { \@@_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`).

```

2767 \keys_define:nn { nicematrix / X-column }
2768 { 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.

```

2769 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2770 {

```

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).

```

2771   \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).

```

2772   \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`.

```

2773   \int_zero_new:N \l_@@_weight_int
2774   \int_set_eq:NN \l_@@_weight_int \c_one_int
2775   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2776   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2777   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2778   {
2779     \@@_error_or_warning:n { negative-weight }
2780     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2781   }
2782   \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).

```

2783 \bool_if:NTF \l_@@_X_columns_aux_bool
2784 {
2785   \@@_make_preamble_ii_iv:nnn
2786   { \l_@@_weight_int \l_@@_X_columns_dim }
2787   { minipage }
2788   { \@@_no_update_width: }
2789 }
2790 {
2791   \tl_gput_right:Nn \g_@@_array_preamble_tl
2792   {
2793     > {
2794       \@@_cell_begin:
2795       \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`.

```

2796 \NotEmpty

```

The following code will nullify the box of the cell.

```

2797 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2798 { \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`.

```

2799 \begin { minipage } { 5 cm } \arraybackslash
2800 {
2801   c
2802   < {
2803     \end { minipage }
2804     \@@_cell_end:
2805   }
2806 }
2807 \int_gincr:N \c@jCol
2808 \@@_rec_preamble_after_col:n
2809 }
2810 }

```

```

2811 \cs_new_protected:Npn \@@_no_update_width:
2812 {
2813   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2814   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2815 }

```

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

```

2816 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2817 {
2818   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2819   { \int_eval:n { \c@jCol + 1 } }
2820   \tl_gput_right:Ne \g_@@_array_preamble_tl
2821   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2822   \@@_rec_preamble:n
2823 }

```

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.

```

2824 \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).

```

2825 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2826 { \@@_fatal:n { Preamble-forgotten } }
2827 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2828 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2829 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2830 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2831 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2832 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox } { @@ _ \token_to_str:N \hline }

```

## 12 The redefinition of `\multicolumn`

The following command must *not* be protected since it begins with `\multispan` (a TeX primitive).

```

2833 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2834 {

```

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

```

2835 \multispan { #1 }
2836 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2837 \begingroup
2838 \bool_if:NT \c_@@_testphase_table_bool
2839 { \tbl_update_multicolumn_cell_data:n { #1 } }
2840 \cs_set_nopar:Npn \@addamp
2841 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2842 \tl_gclear:N \g_@@_preamble_tl
2843 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2844 \exp_args:No \@mkpream \g_@@_preamble_tl
2845 \@addtopreamble \@empty
2846 \endgroup
2847 \bool_if:NT \c_@@_recent_array_bool
2848 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2849 \int_compare:nNnT { #1 } > \c_one_int
2850 {
2851 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2852 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2853 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2854 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2855 {
2856 {
2857 \int_if_zero:nTF \c@jCol
2858 { \int_eval:n { \c@iRow + 1 } }
2859 { \int_use:N \c@iRow }
2860 }
2861 { \int_eval:n { \c@jCol + 1 } }
2862 {
2863 \int_if_zero:nTF \c@jCol
2864 { \int_eval:n { \c@iRow + 1 } }
2865 { \int_use:N \c@iRow }

```

```

2866         }
2867         { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2868         { }
2869     }
2870 }

```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of `colortbl` is available in `\multicolumn`.

```

2871 \RenewDocumentCommand \cellcolor { 0 { } m }
2872 {
2873     \tl_gput_right:Ne \g_@@_pre_code_before_tl
2874     {
2875         \@@_rectanglecolor [ ##1 ]
2876         { \exp_not:n { ##2 } }
2877         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2878         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2879     }
2880     \ignorespaces
2881 }

```

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

```

2882 \cs_set_nopar:Npn \@sharp { #3 }
2883 \@arstrut
2884 \@preamble
2885 \null

```

We add some lines.

```

2886 \int_gadd:Nn \c@jCol { #1 - 1 }
2887 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2888 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2889 \ignorespaces
2890 }

```

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`.

```

2891 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2892 {
2893     \str_case:nnF { #1 }
2894     {
2895         c { \@@_make_m_preamble_i:n #1 }
2896         l { \@@_make_m_preamble_i:n #1 }
2897         r { \@@_make_m_preamble_i:n #1 }
2898         > { \@@_make_m_preamble_ii:nn #1 }
2899         ! { \@@_make_m_preamble_ii:nn #1 }
2900         @ { \@@_make_m_preamble_ii:nn #1 }
2901         | { \@@_make_m_preamble_iii:n #1 }
2902         p { \@@_make_m_preamble_iv:nnn t #1 }
2903         m { \@@_make_m_preamble_iv:nnn c #1 }
2904         b { \@@_make_m_preamble_iv:nnn b #1 }
2905         w { \@@_make_m_preamble_v:nnnn { } #1 }
2906         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2907         \q_stop { }
2908     }
2909     {
2910         \cs_if_exist:cTF { NC @ find @ #1 }
2911         {
2912             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2913             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2914         }
2915         {

```

```

2916         \str_if_eq:nnTF { #1 } { S }
2917         { \@@_fatal:n { unknown~column~type~S } }
2918         { \@@_fatal:nn { unknown~column~type } { #1 } }
2919     }
2920 }
2921 }

```

For c, l and r

```

2922 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2923 {
2924     \tl_gput_right:Nn \g_@@_preamble_tl
2925     {
2926         > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2927         #1
2928         < \@@_cell_end:
2929     }

```

We test for the presence of a <.

```

2930     \@@_make_m_preamble_x:n
2931 }

```

For >, ! and @

```

2932 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2933 {
2934     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2935     \@@_make_m_preamble:n
2936 }

```

For |

```

2937 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2938 {
2939     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2940     \@@_make_m_preamble:n
2941 }

```

For p, m and b

```

2942 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2943 {
2944     \tl_gput_right:Nn \g_@@_preamble_tl
2945     {
2946         > {
2947             \@@_cell_begin:
2948             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2949             \mode_leave_vertical:
2950             \arraybackslash
2951             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2952         }
2953         c
2954         < {
2955             \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2956             \end { minipage }
2957             \@@_cell_end:
2958         }
2959     }

```

We test for the presence of a <.

```

2960     \@@_make_m_preamble_x:n
2961 }

```

For w and W

```

2962 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2963 {
2964     \tl_gput_right:Nn \g_@@_preamble_tl
2965     {

```

```

2966 > {
2967   \dim_set:Nn \l_@@_col_width_dim { #4 }
2968   \hbox_set:Nw \l_@@_cell_box
2969   \@@_cell_begin:
2970   \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2971 }
2972 c
2973 < {
2974   \@@_cell_end:
2975   \hbox_set_end:
2976   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2977   #1
2978   \@@_adjust_size_box:
2979   \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2980 }
2981 }

```

We test for the presence of a <.

```

2982   \@@_make_m_preamble_x:n
2983 }

```

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

```

2984 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2985 {
2986   \str_if_eq:nnTF { #1 } { < }
2987   \@@_make_m_preamble_ix:n
2988   { \@@_make_m_preamble:n { #1 } }
2989 }
2990 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2991 {
2992   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2993   \@@_make_m_preamble_x:n
2994 }

```

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).

```

2995 \cs_new_protected:Npn \@@_put_box_in_flow:
2996 {
2997   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2998   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2999   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3000   { \box_use_drop:N \l_tmpa_box }
3001   \@@_put_box_in_flow_i:
3002 }

```

The command \@@\_put\_box\_in\_flow\_i: is used when the value of \l\_@@\_baseline\_tl is different of c (the initial value).

```

3003 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3004 {
3005   \pgfpicture
3006   \@@_qpoint:n { row - 1 }
3007   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3008   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3009   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3010   \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).

```

3011   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3012   {

```

```

3013     \int_set:Nn \l_tmpa_int
3014     {
3015         \str_range:Nnn
3016         \l_@@_baseline_tl
3017         6
3018         { \tl_count:o \l_@@_baseline_tl }
3019     }
3020     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3021 }
3022 {
3023     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3024     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3025     {
3026         \str_if_eq:onTF \l_@@_baseline_tl { b }
3027         { \int_set_eq:NN \l_tmpa_int \c_iRow }
3028         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3029     }
3030     \bool_lazy_or:nnT
3031     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3032     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3033     {
3034         \@@_error:n { bad-value-for-baseline }
3035         \int_set_eq:NN \l_tmpa_int \c_one_int
3036     }
3037     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3038     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3039 }
3040 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3041     \endpgfpicture
3042     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3043     \box_use_drop:N \l_tmpa_box
3044 }

```

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).

```

3045 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3046 {

```

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.

```

3047     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3048     {
3049         \int_compare:nNnT \c_jCol > \c_one_int
3050         {
3051             \box_set_wd:Nn \l_@@_the_array_box
3052             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3053         }
3054     }

```

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

```

3055     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3056     \bool_if:NT \l_@@_caption_above_bool
3057     {
3058         \tl_if_empty:NF \l_@@_caption_tl
3059         {

```

```

3060      \bool_set_false:N \g_@@_caption_finished_bool
3061      \int_gzero:N \c@tabularnote
3062      \@@_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).

```

3063      \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3064      {
3065        \tl_gput_right:Ne \g_@@_aux_tl
3066        {
3067          \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3068            { \int_use:N \g_@@_notes_caption_int }
3069        }
3070        \int_gzero:N \g_@@_notes_caption_int
3071      }
3072    }
3073  }

```

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

```

3074    \hbox
3075    {
3076      \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.

```

3077      \@@_create_extra_nodes:
3078      \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3079    }

```

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

```

3080    \bool_lazy_any:nT
3081    {
3082      { ! \seq_if_empty_p:N \g_@@_notes_seq }
3083      { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3084      { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3085    }
3086    \@@_insert_tabularnotes:
3087    \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3088    \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3089    \end { minipage }
3090  }

```

```

3091  \cs_new_protected:Npn \@@_insert_caption:
3092  {
3093    \tl_if_empty:NF \l_@@_caption_tl
3094    {
3095      \cs_if_exist:NTF \@capttype
3096      { \@@_insert_caption_i: }
3097      { \@@_error:n { caption-outside-float } }
3098    }
3099  }

```

```

3100  \cs_new_protected:Npn \@@_insert_caption_i:
3101  {
3102    \group_begin:

```

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

```
3103 \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.

```
3104 \IfPackageLoadedT { floatrow }
3105 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3106 \tl_if_empty:NTF \l_@@_short_caption_tl
3107 { \caption }
3108 { \caption [ \l_@@_short_caption_tl ] }
3109 { \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.

```
3110 \bool_if:NF \g_@@_caption_finished_bool
3111 {
3112   \bool_gset_true:N \g_@@_caption_finished_bool
3113   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3114   \int_gzero:N \c@tabularnote
3115 }
3116 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3117 \group_end:
3118 }

3119 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3120 {
3121   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3122   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3123 }

3124 \cs_new_protected:Npn \@@_insert_tabularnotes:
3125 {
3126   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3127   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3128   \skip_vertical:N 0.65ex
```

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

```
3129 \group_begin:
3130 \l_@@_notes_code_before_tl
3131 \tl_if_empty:NF \g_@@_tabularnote_tl
3132 {
3133   \g_@@_tabularnote_tl \par
3134   \tl_gclear:N \g_@@_tabularnote_tl
3135 }
```

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.

```
3136 \int_compare:nNnT \c@tabularnote > \c_zero_int
3137 {
3138   \bool_if:NTF \l_@@_notes_para_bool
3139   {
3140     \begin { tabularnotes* }
3141     \seq_map_inline:Nn \g_@@_notes_seq
3142     { \@@_one_tabularnote:nn ##1 }
3143     \strut
3144     \end { tabularnotes* }
3145 }
```

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

```
3145 \par
```



```

3146     }
3147     {
3148         \tabularnotes
3149         \seq_map_inline:Nn \g_@@_notes_seq
3150         { \@@_one_tabularnote:nn ##1 }
3151         \strut
3152         \endtabularnotes
3153     }
3154 }
3155 \unskip
3156 \group_end:
3157 \bool_if:NT \l_@@_notes_bottomrule_bool
3158 {
3159     \IfPackageLoadedTF { booktabs }
3160     {

```

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

```

3161         \skip_vertical:N \aboverulesep

```

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

```

3162         { \CT@arc@ \hrule height \heavyrulewidth }
3163     }
3164     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3165 }
3166 \l_@@_notes_code_after_tl
3167 \seq_gclear:N \g_@@_notes_seq
3168 \seq_gclear:N \g_@@_notes_in_caption_seq
3169 \int_gzero:N \c@tabularnote
3170 }

```

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.

```

3171 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3172 {
3173     \tl_if_novalue:nTF { #1 }
3174     { \item }
3175     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3176 }

```

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`.

```

3177 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3178 {
3179     \pgfpicture
3180     \@@_qpoint:n { row - 1 }
3181     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3182     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3183     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3184     \endpgfpicture
3185     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3186     \int_if_zero:nT \l_@@_first_row_int
3187     {
3188         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3189         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3190     }
3191     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3192 }

```

Now, the general case.

```

3193 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3194 {

```

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

```

3195 \str_if_eq:eeT \l_@@_baseline_tl { t }
3196 { \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`.

```

3197 \pgfpicture
3198 \@@_qpoint:n { row - 1 }
3199 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3200 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3201 {
3202   \int_set:Nn \l_tmpa_int
3203   {
3204     \str_range:Nnn
3205     \l_@@_baseline_tl
3206     6
3207     { \tl_count:o \l_@@_baseline_tl }
3208   }
3209   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3210 }
3211 {
3212   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3213   \bool_lazy_or:nnT
3214   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3215   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3216   {
3217     \@@_error:n { bad-value-for~baseline }
3218     \int_set:Nn \l_tmpa_int 1
3219   }
3220   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3221 }
3222 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3223 \endpgfpicture
3224 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3225 \int_if_zero:nT \l_@@_first_row_int
3226 {
3227   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3228   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3229 }
3230 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3231 }

```

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.

```

3232 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3233 {

```

We will compute the real width of both delimiters used.

```

3234 \dim_zero_new:N \l_@@_real_left_delim_dim
3235 \dim_zero_new:N \l_@@_real_right_delim_dim
3236 \hbox_set:Nn \l_tmpb_box
3237 {
3238   \m@th % added 2024/11/21
3239   \c_math_toggle_token
3240   \left #1
3241   \vcenter
3242   {
3243     \vbox_to_ht:nn
3244     { \box_ht_plus_dp:N \l_tmpa_box }
3245     { }
3246   }
3247   \right .

```

```

3248     \c_math_toggle_token
3249   }
3250   \dim_set:Nn \l_@@_real_left_delim_dim
3251   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3252   \hbox_set:Nn \l_tmpb_box
3253   {
3254     \m@th % added 2024/11/21
3255     \c_math_toggle_token
3256     \left .
3257     \vbox_to_ht:nn
3258       { \box_ht_plus_dp:N \l_tmpa_box }
3259       { }
3260     \right #2
3261     \c_math_toggle_token
3262   }
3263   \dim_set:Nn \l_@@_real_right_delim_dim
3264   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3265   \skip_horizontal:N \l_@@_left_delim_dim
3266   \skip_horizontal:N -\l_@@_real_left_delim_dim
3267   \@@_put_box_in_flow:
3268   \skip_horizontal:N \l_@@_right_delim_dim
3269   \skip_horizontal:N -\l_@@_real_right_delim_dim
3270 }

```

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).

```

3271 \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).

```

3272 {
3273   \peek_remove_spaces:n
3274   {
3275     \peek_meaning:NTF \end
3276     \@@_analyze_end:Nn
3277     {
3278       \@@_transform_preamble:

```

Here is the call to `\array` (we have a dedicated macro `\@@_array:n` because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3279     \@@_array:o \g_@@_array_preamble_tl
3280   }
3281 }
3282 }
3283 {
3284   \@@_create_col_nodes:
3285   \endarray
3286 }

```

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

```

3287 \NewDocumentEnvironment { @@-light-syntax } { b }
3288 {

```

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.

```

3289 \tl_if_empty:nT { #1 }
3290 { \@@_fatal:n { empty~environment } }
3291 \tl_if_in:nnT { #1 } { & }
3292 { \@@_fatal:n { ampersand~in~light-syntax } }
3293 \tl_if_in:nnT { #1 } { \ }
3294 { \@@_fatal:n { double-backslash~in~light-syntax } }

```

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`.

```

3295 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3296 }

```

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.

```

3297 {
3298 \@@_create_col_nodes:
3299 \endarray
3300 }
3301 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3302 {
3303 \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).

```

3304 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3305 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3306 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3307 \seq_set_split:Nee
3308 \seq_set_split:Non
3309 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3310 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3311 \tl_if_empty:NF \l_tmpa_tl
3312 { \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.

```

3313 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3314 { \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`).

```

3315 \tl_build_begin:N \l_@@_new_body_tl
3316 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3317 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3318 \@@_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).

```

3319 \seq_map_inline:Nn \l_@@_rows_seq
3320 {
3321   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3322   \@@_line_with_light_syntax:n { ##1 }
3323 }
3324 \tl_build_end:N \l_@@_new_body_tl
3325 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3326 {
3327   \int_set:Nn \l_@@_last_col_int
3328   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3329 }

```

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.

```

3330 \@@_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`).

```

3331 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3332 }
3333 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3334 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3335 {
3336   \seq_clear_new:N \l_@@_cells_seq
3337   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3338   \int_set:Nn \l_@@_nb_cols_int
3339   {
3340     \int_max:nn
3341     \l_@@_nb_cols_int
3342     { \seq_count:N \l_@@_cells_seq }
3343   }
3344   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3345   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3346   \seq_map_inline:Nn \l_@@_cells_seq
3347   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3348 }

```

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`.

```

3349 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3350 {
3351   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3352   { \@@_fatal:n { empty-environment } }

```

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

```

3353 \end { #2 }
3354 }

```

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 such as `columns-width`).

```

3355 \cs_new:Npn \@@_create_col_nodes:
3356 {
3357   \crrc
3358   \int_if_zero:nT \l_@@_first_col_int
3359   {
3360     \omit

```

```

3361 \hbox_overlap_left:n
3362 {
3363   \bool_if:NT \l_@@_code_before_bool
3364   { \pgfsys@markposition { \@@_env: - col - 0 } }
3365   \pgfpicture
3366   \pgfrememberpicturepositiononpagetrue
3367   \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3368   \str_if_empty:NF \l_@@_name_str
3369   { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3370   \endpgfpicture
3371   \skip_horizontal:N 2\col@sep
3372   \skip_horizontal:N \g_@@_width_first_col_dim
3373 }
3374 &
3375 }
3376 \omit

```

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

```

3377 \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`).

```

3378 \int_if_zero:nTF \l_@@_first_col_int
3379 {
3380   \bool_if:NT \l_@@_code_before_bool
3381   {
3382     \hbox
3383     {
3384       \skip_horizontal:N -0.5\arrayrulewidth
3385       \pgfsys@markposition { \@@_env: - col - 1 }
3386       \skip_horizontal:N 0.5\arrayrulewidth
3387     }
3388   }
3389   \pgfpicture
3390   \pgfrememberpicturepositiononpagetrue
3391   \pgfcoordinate { \@@_env: - col - 1 }
3392   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3393   \str_if_empty:NF \l_@@_name_str
3394   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3395   \endpgfpicture
3396 }
3397 {
3398   \bool_if:NT \l_@@_code_before_bool
3399   {
3400     \hbox
3401     {
3402       \skip_horizontal:N 0.5\arrayrulewidth
3403       \pgfsys@markposition { \@@_env: - col - 1 }
3404       \skip_horizontal:N -0.5\arrayrulewidth
3405     }
3406   }
3407   \pgfpicture
3408   \pgfrememberpicturepositiononpagetrue
3409   \pgfcoordinate { \@@_env: - col - 1 }
3410   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3411   \str_if_empty:NF \l_@@_name_str
3412   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3413   \endpgfpicture
3414 }

```

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.

```

3415 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3416 \bool_if:NF \l_@@_auto_columns_width_bool
3417 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3418 {
3419   \bool_lazy_and:nnTF
3420     \l_@@_auto_columns_width_bool
3421     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3422     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3423     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3424     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3425 }
3426 \skip_horizontal:N \g_tmpa_skip
3427 \hbox
3428 {
3429   \bool_if:NT \l_@@_code_before_bool
3430   {
3431     \hbox
3432     {
3433       \skip_horizontal:N -0.5\arrayrulewidth
3434       \pgfsys@markposition { \@@_env: - col - 2 }
3435       \skip_horizontal:N 0.5\arrayrulewidth
3436     }
3437   }
3438   \pgfpicture
3439   \pgfrememberpicturerepositiononpagetrue
3440   \pgfcoordinate { \@@_env: - col - 2 }
3441   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3442   \str_if_empty:NF \l_@@_name_str
3443   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3444   \endpgfpicture
3445 }

```

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.

```

3446 \int_gset_eq:NN \g_tmpa_int \c_one_int
3447 \bool_if:NTF \g_@@_last_col_found_bool
3448 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3449 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3450 {
3451   &
3452   \omit
3453   \int_gincr:N \g_tmpa_int

```

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

```

3454 \skip_horizontal:N \g_tmpa_skip
3455 \bool_if:NT \l_@@_code_before_bool
3456 {
3457   \hbox
3458   {
3459     \skip_horizontal:N -0.5\arrayrulewidth
3460     \pgfsys@markposition
3461     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3462     \skip_horizontal:N 0.5\arrayrulewidth
3463   }
3464 }

```

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

```

3465 \pgfpicture
3466 \pgfrememberpicturerepositiononpagetrue
3467 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3468 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3469 \str_if_empty:NF \l_@@_name_str

```

```

3470     {
3471         \pgfnodealias
3472         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3473         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3474     }
3475 \endpgfpicture
3476 }

3477 &
3478 \omit

```

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

```

3479 \int_if_zero:nT \g_@@_col_total_int
3480 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3481 \skip_horizontal:N \g_tmpa_skip
3482 \int_gincr:N \g_tmpa_int
3483 \bool_lazy_any:nF
3484 {
3485     \g_@@_delims_bool
3486     \l_@@_tabular_bool
3487     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3488     \l_@@_exterior_arraycolsep_bool
3489     \l_@@_bar_at_end_of_pream_bool
3490 }
3491 { \skip_horizontal:N -\col@sep }
3492 \bool_if:NT \l_@@_code_before_bool
3493 {
3494     \hbox
3495     {
3496         \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.

```

3497     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3498     { \skip_horizontal:N -\arraycolsep }
3499     \pgfsys@markposition
3500     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3501     \skip_horizontal:N 0.5\arrayrulewidth
3502     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3503     { \skip_horizontal:N \arraycolsep }
3504 }
3505 }
3506 \pgfpicture
3507 \pgfrememberpicturepositiononpagetrue
3508 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3509 {
3510     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3511     {
3512         \pgfpoint
3513         { - 0.5 \arrayrulewidth - \arraycolsep }
3514         \c_zero_dim
3515     }
3516     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3517 }
3518 \str_if_empty:NF \l_@@_name_str
3519 {
3520     \pgfnodealias
3521     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3522     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3523 }
3524 \endpgfpicture

```



```

3525 \bool_if:NT \g_@@_last_col_found_bool
3526 {
3527   \hbox_overlap_right:n
3528   {
3529     \skip_horizontal:N \g_@@_width_last_col_dim
3530     \skip_horizontal:N \col@sep
3531     \bool_if:NT \l_@@_code_before_bool
3532     {
3533       \pgfsys@markposition
3534       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3535     }
3536     \pgfpicture
3537     \pgfrememberpicturepositiononpagetrue
3538     \pgfcoordinate
3539     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3540     \pgfpointorigin
3541     \str_if_empty:NF \l_@@_name_str
3542     {
3543       \pgfnodealias
3544       {
3545         \l_@@_name_str - col
3546         - \int_eval:n { \g_@@_col_total_int + 1 }
3547       }
3548       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3549     }
3550     \endpgfpicture
3551   }
3552 }
3553 % \cr
3554 }

```

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

```

3555 \tl_const:Nn \c_@@_preamble_first_col_tl
3556 {
3557   >
3558   {

```

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

```

3559     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3560     \bool_gset_true:N \g_@@_after_col_zero_bool
3561     \@@_begin_of_row:
3562     \hbox_set:Nw \l_@@_cell_box
3563     \@@_math_toggle:
3564     \@@_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”.

```

3565     \int_compare:nNnT \c@iRow > \c_zero_int
3566     {
3567       \bool_lazy_or:nnT
3568       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3569       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3570       {
3571         \l_@@_code_for_first_col_tl
3572         \xglobal \colorlet { nicematrix-first-col } { . }
3573       }
3574     }
3575 }

```

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`.

```

3576   1

```

```

3577 <
3578 {
3579   \@@_math_toggle:
3580   \hbox_set_end:
3581   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3582   \@@_adjust_size_box:
3583   \@@_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.

```

3584   \dim_gset:Nn \g_@@_width_first_col_dim
3585   { \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.

```

3586   \hbox_overlap_left:n
3587   {
3588     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3589     \@@_node_for_cell:
3590     { \box_use_drop:N \l_@@_cell_box }
3591     \skip_horizontal:N \l_@@_left_delim_dim
3592     \skip_horizontal:N \l_@@_left_margin_dim
3593     \skip_horizontal:N \l_@@_extra_left_margin_dim
3594   }
3595   \bool_gset_false:N \g_@@_empty_cell_bool
3596   \skip_horizontal:N -2\col@sep
3597 }
3598 }

```

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

```

3599 \tl_const:Nn \c_@@_preamble_last_col_tl
3600 {
3601   >
3602   {
3603     \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).

```

3604   \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.

```

3605   \bool_gset_true:N \g_@@_last_col_found_bool
3606   \int_gincr:N \c@jCol
3607   \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3608   \hbox_set:Nw \l_@@_cell_box
3609   \@@_math_toggle:
3610   \@@_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”.

```

3611   \int_compare:nNnT \c@iRow > \c_zero_int
3612   {
3613     \bool_lazy_or:nnT
3614     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3615     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3616     {
3617       \l_@@_code_for_last_col_tl
3618       \xglobal \colorlet { nicematrix-last-col } { . }
3619     }
3620   }
3621 }
3622 1
3623 <
3624 {
3625   \@@_math_toggle:
3626   \hbox_set_end:
3627   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

```

3628 \@@_adjust_size_box:
3629 \@@_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.

```

3630 \dim_gset:Nn \g_@@_width_last_col_dim
3631 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3632 \skip_horizontal:N -2\col@sep

```

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

```

3633 \hbox_overlap_right:n
3634 {
3635   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3636   {
3637     \skip_horizontal:N \l_@@_right_delim_dim
3638     \skip_horizontal:N \l_@@_right_margin_dim
3639     \skip_horizontal:N \l_@@_extra_right_margin_dim
3640     \@@_node_for_cell:
3641   }
3642 }
3643 \bool_gset_false:N \g_@@_empty_cell_bool
3644 }
3645 }

```

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

```

3646 \NewDocumentEnvironment { NiceArray } { }
3647 {
3648   \bool_gset_false:N \g_@@_delims_bool
3649   \str_if_empty:NT \g_@@_name_env_str
3650   { \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).

```

3651 \NiceArrayWithDelims . .
3652 }
3653 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3654 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3655 {
3656   \NewDocumentEnvironment { #1 NiceArray } { }
3657   {
3658     \bool_gset_true:N \g_@@_delims_bool
3659     \str_if_empty:NT \g_@@_name_env_str
3660     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3661     \@@_test_if_math_mode:
3662     \NiceArrayWithDelims #2 #3
3663   }
3664   { \endNiceArrayWithDelims }
3665 }
3666 \@@_def_env:nnn p ( )
3667 \@@_def_env:nnn b [ ]
3668 \@@_def_env:nnn B \{ \}
3669 \@@_def_env:nnn v | |
3670 \@@_def_env:nnn V \l \r

```

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

```

3671 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3672 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3673 {
3674   \bool_set_false:N \l_@@_preamble_bool
3675   \tl_clear:N \l_tmpa_tl
3676   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3677     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3678   \tl_put_right:Nn \l_tmpa_tl
3679     {
3680       *
3681       {
3682         \int_case:nnF \l_@@_last_col_int
3683         {
3684           { -2 } { \c@MaxMatrixCols }
3685           { -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).

```

3686       }
3687       { \int_eval:n { \l_@@_last_col_int - 1 } }
3688     }
3689     { #2 }
3690   }
3691   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3692   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3693 }
3694 \clist_map_inline:nn { p , b , B , v , V }
3695 {
3696   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3697   {
3698     \bool_gset_true:N \g_@@_delims_bool
3699     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3700     \int_if_zero:nT \l_@@_last_col_int
3701     {
3702       \bool_set_true:N \l_@@_last_col_without_value_bool
3703       \int_set:Nn \l_@@_last_col_int { -1 }
3704     }
3705     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3706     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3707   }
3708   { \use:c { end #1 NiceArray } }
3709 }

```

We define also an environment `{NiceMatrix}`

```

3710 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3711 {
3712   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3713   \int_if_zero:nT \l_@@_last_col_int
3714   {
3715     \bool_set_true:N \l_@@_last_col_without_value_bool
3716     \int_set:Nn \l_@@_last_col_int { -1 }
3717   }
3718   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3719   \bool_lazy_or:nnT
3720     { \clist_if_empty_p:N \l_@@_vlines_clist }
3721     { \l_@@_except_borders_bool }
3722     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3723   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3724 }
3725 { \endNiceArray }

```

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

```
3726 \cs_new_protected:Npn \@@_NotEmpty:
3727 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

## 14 {NiceTabular}, {NiceTabularX} and {NiceTabular\*}

```
3728 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3729 {
```

If the dimension `\l_@@_width_dim` is equal to 0 pt, that means that it has not been set by a previous use of `\NiceMatrixOptions`.

```
3730   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3731     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3732   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3733   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3734   \tl_if_empty:NF \l_@@_short_caption_tl
3735   {
3736     \tl_if_empty:NT \l_@@_caption_tl
3737     {
3738       \@@_error_or_warning:n { short-caption-without-caption }
3739       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3740     }
3741   }
3742   \tl_if_empty:NF \l_@@_label_tl
3743   {
3744     \tl_if_empty:NT \l_@@_caption_tl
3745     { \@@_error_or_warning:n { label-without-caption } }
3746   }
3747   \NewDocumentEnvironment { TabularNote } { b }
3748   {
3749     \bool_if:NTF \l_@@_in_code_after_bool
3750     { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3751     {
3752       \tl_if_empty:NF \g_@@_tabularnote_tl
3753       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3754       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3755     }
3756   }
3757   { }
3758   \@@_settings_for_tabular:
3759   \NiceArray { #2 }
3760 }
3761 { \endNiceArray }
3762 \cs_new_protected:Npn \@@_settings_for_tabular:
3763 {
3764   \bool_set_true:N \l_@@_tabular_bool
3765   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3766   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3767   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3768 }
```

```
3769 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3770 {
3771   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3772   \dim_zero_new:N \l_@@_width_dim
3773   \dim_set:Nn \l_@@_width_dim { #1 }
3774   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3775   \@@_settings_for_tabular:
3776   \NiceArray { #3 }
3777 }
3778 {
3779   \endNiceArray
```

```

3780 \int_if_zero:nT \g_@@_total_X_weight_int
3781 { \@@_error:n { NiceTabularX~without~X } }
3782 }

3783 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3784 {
3785   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3786   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3787   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3788   \@@_settings_for_tabular:
3789   \NiceArray { #3 }
3790 }
3791 { \endNiceArray }

```

## 15 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`).

```

3792 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3793 {
3794   \bool_lazy_all:nT
3795   {
3796     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3797     \l_@@_hvlines_bool
3798     { ! \g_@@_delims_bool }
3799     { ! \l_@@_except_borders_bool }
3800   }
3801   {
3802     \bool_set_true:N \l_@@_except_borders_bool
3803     \clist_if_empty:NF \l_@@_corners_clist
3804     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3805     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3806     {
3807       \@@_stroke_block:nnn
3808       {
3809         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3810         draw = \l_@@_rules_color_tl
3811       }
3812       { 1-1 }
3813       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3814     }
3815   }
3816 }

3817 \cs_new_protected:Npn \@@_after_array:
3818 {

```

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`.

```

3819 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3820 \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.

```
3821 \bool_if:NT \g_@@_last_col_found_bool
3822 { \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`.

```
3823 \bool_if:NT \l_@@_last_col_without_value_bool
3824 { \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.

```
3825 \bool_if:NT \l_@@_last_row_without_value_bool
3826 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3827 \tl_gput_right:Ne \g_@@_aux_tl
3828 {
3829   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3830   {
3831     \int_use:N \l_@@_first_row_int ,
3832     \int_use:N \c@iRow ,
3833     \int_use:N \g_@@_row_total_int ,
3834     \int_use:N \l_@@_first_col_int ,
3835     \int_use:N \c@jCol ,
3836     \int_use:N \g_@@_col_total_int
3837   }
3838 }
```

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`).

```
3839 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3840 {
3841   \tl_gput_right:Ne \g_@@_aux_tl
3842   {
3843     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3844     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3845   }
3846 }
3847 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3848 {
3849   \tl_gput_right:Ne \g_@@_aux_tl
3850   {
3851     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3852     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3853     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3854     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3855   }
3856 }
```

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

```
3857 \@@_create_diag_nodes:
```

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

```
3858 \pgfpicture
3859 \int_step_inline:nn \c@iRow
3860 {
3861   \pgfnodealias
3862   { \@@_env: - ##1 - last }
3863   { \@@_env: - ##1 - \int_use:N \c@jCol }
3864 }
3865 \int_step_inline:nn \c@jCol
3866 {
3867   \pgfnodealias
```

```

3868         { \@@_env: - last - ##1 }
3869         { \@@_env: - \int_use:N \c@iRow - ##1 }
3870     }
3871     \str_if_empty:NF \l_@@_name_str
3872     {
3873         \int_step_inline:nn \c@iRow
3874         {
3875             \pgfnodealias
3876             { \l_@@_name_str - ##1 - last }
3877             { \@@_env: - ##1 - \int_use:N \c@jCol }
3878         }
3879         \int_step_inline:nn \c@jCol
3880         {
3881             \pgfnodealias
3882             { \l_@@_name_str - last - ##1 }
3883             { \@@_env: - \int_use:N \c@iRow - ##1 }
3884         }
3885     }
3886     \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.

```

3887     \bool_if:NT \l_@@_parallelize_diags_bool
3888     {
3889         \int_gzero_new:N \g_@@_ddots_int
3890         \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.

```

3891         \dim_gzero_new:N \g_@@_delta_x_one_dim
3892         \dim_gzero_new:N \g_@@_delta_y_one_dim
3893         \dim_gzero_new:N \g_@@_delta_x_two_dim
3894         \dim_gzero_new:N \g_@@_delta_y_two_dim
3895     }
3896     \int_zero_new:N \l_@@_initial_i_int
3897     \int_zero_new:N \l_@@_initial_j_int
3898     \int_zero_new:N \l_@@_final_i_int
3899     \int_zero_new:N \l_@@_final_j_int
3900     \bool_set_false:N \l_@@_initial_open_bool
3901     \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.

```

3902     \bool_if:NT \l_@@_small_bool
3903     {
3904         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3905         \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.

```

3906         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3907         { 0.6 \l_@@_xdots_shorten_start_dim }
3908         \dim_set:Nn \l_@@_xdots_shorten_end_dim
3909         { 0.6 \l_@@_xdots_shorten_end_dim }
3910     }

```

---

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



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

```
3911 \@@_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_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```
3912 \clist_if_empty:NF \l_@@_corners_clist
3913 {
3914   \bool_if:NTF \l_@@_no_cell_nodes_bool
3915   { \@@_error:n { corners~with~no~cell~nodes } }
3916   { \@@_compute_corners: }
3917 }
```

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

```
3918 \@@_adjust_pos_of_blocks_seq:
3919 \@@_deal_with_rounded_corners:
3920 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3921 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3922 \IfPackageLoadedT { tikz }
3923 {
3924   \tikzset
3925   {
3926     every-picture / .style =
3927     {
3928       overlay ,
3929       remember-picture ,
3930       name-prefix = \@@_env: -
3931     }
3932   }
3933 }
3934 \bool_if:NT \c_@@_recent_array_bool
3935 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3936 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3937 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3938 \cs_set_eq:NN \OverBrace \@@_OverBrace
3939 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3940 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3941 \cs_set_eq:NN \line \@@_line
3942 \g_@@_pre_code_after_tl
3943 \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 `\CodeAfter` to be *no-op* now.

```
3944 \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).

```
3945 \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`).

```
3946 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3947 { \@@_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:`.

```

3948 \bool_set_true:N \l_@@_in_code_after_bool
3949 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3950 \scan_stop:
3951 \tl_gclear:N \g_nicematrix_code_after_tl
3952 \group_end:

```

`\g_@@_pre_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor`. These instructions will be written on the aux file to be added to the `code-before` in the next run.

```

3953 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3954 \tl_if_empty:NF \g_@@_pre_code_before_tl
3955 {
3956   \tl_gput_right:Ne \g_@@_aux_tl
3957   {
3958     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3959     { \exp_not:o \g_@@_pre_code_before_tl }
3960   }
3961   \tl_gclear:N \g_@@_pre_code_before_tl
3962 }
3963 \tl_if_empty:NF \g_nicematrix_code_before_tl
3964 {
3965   \tl_gput_right:Ne \g_@@_aux_tl
3966   {
3967     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3968     { \exp_not:o \g_nicematrix_code_before_tl }
3969   }
3970   \tl_gclear:N \g_nicematrix_code_before_tl
3971 }

3972 \str_gclear:N \g_@@_name_env_str
3973 \@@_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`.

```

3974 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3975 }

```

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`.

```

3976 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3977 { \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).

```

3978 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3979 {

```

---

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

```

3980 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3981 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3982 }

```

The following command must *not* be protected.

```

3983 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3984 {
3985   { #1 }
3986   { #2 }
3987   {
3988     \int_compare:nNnTF { #3 } > { 98 }
3989     { \int_use:N \c@iRow }
3990     { #3 }
3991   }
3992   {
3993     \int_compare:nNnTF { #4 } > { 98 }
3994     { \int_use:N \c@jCol }
3995     { #4 }
3996   }
3997   { #5 }
3998 }

```

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).

```

3999 \hook_gput_code:nnn { begindocument } { . }
4000 {
4001   \cs_new_protected:Npe \@@_draw_dotted_lines:
4002   {
4003     \c_@@_pgfortikzpicture_tl
4004     \@@_draw_dotted_lines_i:
4005     \c_@@_endpgfortikzpicture_tl
4006   }
4007 }

```

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

```

4008 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4009 {
4010   \pgfrememberpicturerepositiononpagetrue
4011   \pgf@relevantforpicturesizefalse
4012   \g_@@_HVdotsfor_lines_tl
4013   \g_@@_Vdots_lines_tl
4014   \g_@@_Ddots_lines_tl
4015   \g_@@_Idots_lines_tl
4016   \g_@@_Cdots_lines_tl
4017   \g_@@_Ldots_lines_tl
4018 }

4019 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4020 {
4021   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4022   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4023 }

```

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

```

4024 \pgfdeclareshape { @@_diag_node }
4025 {
4026   \savedanchor { \five }
4027   {
4028     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

4029     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4030   }
4031   \anchor { 5 } { \five }
4032   \anchor { center } { \pgfpointorigin }
4033   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4034   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4035   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4036   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4037   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4038   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4039   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4040   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4041   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4042   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4043 }

```

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).

```

4044 \cs_new_protected:Npn \@@_create_diag_nodes:
4045 {
4046   \pgfpicture
4047   \pgfrememberpicturepositiononpagetrue
4048   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4049   {
4050     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4051     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4052     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4053     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4054     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4055     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4056     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4057     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4058     \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.

```

4059     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4060     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4061     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4062     \str_if_empty:NF \l_@@_name_str
4063     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4064   }

```

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

```

4065     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4066     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4067     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4068     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4069     \pgfcoordinate
4070     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4071     \pgfnodealias
4072     { \@@_env: - last }
4073     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4074     \str_if_empty:NF \l_@@_name_str
4075     {
4076       \pgfnodealias
4077       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4078       { \@@_env: - \int_use:N \l_tmpa_int }
4079       \pgfnodealias
4080       { \l_@@_name_str - last }
4081       { \@@_env: - last }
4082     }

```

```

4083 \endpgfpicture
4084 }

```

## 16 We draw the dotted lines

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

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

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

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

This command computes:

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

```

4085 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4086 {

```

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

```

4087 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4088 \int_set:Nn \l_@@_initial_i_int { #1 }
4089 \int_set:Nn \l_@@_initial_j_int { #2 }
4090 \int_set:Nn \l_@@_final_i_int { #1 }
4091 \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.

```

4092 \bool_set_false:N \l_@@_stop_loop_bool
4093 \bool_do_until:Nn \l_@@_stop_loop_bool
4094 {
4095   \int_add:Nn \l_@@_final_i_int { #3 }
4096   \int_add:Nn \l_@@_final_j_int { #4 }
4097   \bool_set_false:N \l_@@_final_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4098     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4099         \if_int_compare:w #3 = \c_one_int
4100             \bool_set_true:N \l_@@_final_open_bool
4101         \else:
4102             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4103                 \bool_set_true:N \l_@@_final_open_bool
4104             \fi:
4105         \fi:
4106     \else:
4107         \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4108             \if_int_compare:w #4 = -1
4109                 \bool_set_true:N \l_@@_final_open_bool
4110             \fi:
4111         \else:
4112             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4113                 \if_int_compare:w #4 = \c_one_int
4114                     \bool_set_true:N \l_@@_final_open_bool
4115                 \fi:
4116             \fi:
4117         \fi:
4118     \fi:
4119     \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.

```

4120     {

```

We do a step backwards.

```

4121         \int_sub:Nn \l_@@_final_i_int { #3 }
4122         \int_sub:Nn \l_@@_final_j_int { #4 }
4123         \bool_set_true:N \l_@@_stop_loop_bool
4124     }

```

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`.

```

4125     {
4126         \cs_if_exist:cTF
4127         {
4128             @@ _ dotted _
4129             \int_use:N \l_@@_final_i_int -
4130             \int_use:N \l_@@_final_j_int
4131         }
4132         {
4133             \int_sub:Nn \l_@@_final_i_int { #3 }
4134             \int_sub:Nn \l_@@_final_j_int { #4 }
4135             \bool_set_true:N \l_@@_final_open_bool
4136             \bool_set_true:N \l_@@_stop_loop_bool
4137         }
4138         {
4139             \cs_if_exist:cTF
4140             {
4141                 pgf @ sh @ ns @ \@@_env:
4142                 - \int_use:N \l_@@_final_i_int
4143                 - \int_use:N \l_@@_final_j_int
4144             }
4145             { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4146     {

```

```

4147         \cs_set_nopar:cpn
4148         {
4149             @@ _ dotted _
4150             \int_use:N \l_@@_final_i_int -
4151             \int_use:N \l_@@_final_j_int
4152         }
4153         { }
4154     }
4155 }
4156 }
4157 }

```

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

```

4158     \bool_set_false:N \l_@@_stop_loop_bool

```

The following line of code is only for efficiency in the following loop.

```

4159     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4160     \bool_do_until:Nn \l_@@_stop_loop_bool
4161     {
4162         \int_sub:Nn \l_@@_initial_i_int { #3 }
4163         \int_sub:Nn \l_@@_initial_j_int { #4 }
4164         \bool_set_false:N \l_@@_initial_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4165         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4166         \if_int_compare:w #3 = \c_one_int
4167         \bool_set_true:N \l_@@_initial_open_bool
4168         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4169         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4170         \bool_set_true:N \l_@@_initial_open_bool
4171         \fi:
4172         \fi:
4173     \else:
4174         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4175         \if_int_compare:w #4 = \c_one_int
4176         \bool_set_true:N \l_@@_initial_open_bool
4177         \fi:
4178         \else:
4179             \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4180             \if_int_compare:w #4 = -1
4181             \bool_set_true:N \l_@@_initial_open_bool
4182             \fi:
4183             \fi:
4184             \fi:
4185         \fi:
4186     \bool_if:NTF \l_@@_initial_open_bool
4187     {
4188         \int_add:Nn \l_@@_initial_i_int { #3 }
4189         \int_add:Nn \l_@@_initial_j_int { #4 }
4190         \bool_set_true:N \l_@@_stop_loop_bool
4191     }
4192     {
4193         \cs_if_exist:cTF
4194         {
4195             @@ _ dotted _
4196             \int_use:N \l_@@_initial_i_int -
4197             \int_use:N \l_@@_initial_j_int
4198         }

```

```

4199         {
4200             \int_add:Nn \l_@@_initial_i_int { #3 }
4201             \int_add:Nn \l_@@_initial_j_int { #4 }
4202             \bool_set_true:N \l_@@_initial_open_bool
4203             \bool_set_true:N \l_@@_stop_loop_bool
4204         }
4205         {
4206             \cs_if_exist:cTF
4207             {
4208                 pgf @ sh @ ns @ \@@_env:
4209                 - \int_use:N \l_@@_initial_i_int
4210                 - \int_use:N \l_@@_initial_j_int
4211             }
4212             { \bool_set_true:N \l_@@_stop_loop_bool }
4213         {
4214             \cs_set_nopar:cpn
4215             {
4216                 @@ _ dotted _
4217                 \int_use:N \l_@@_initial_i_int -
4218                 \int_use:N \l_@@_initial_j_int
4219             }
4220             { }
4221         }
4222     }
4223 }
4224 }

```

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.

```

4225     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4226     {
4227         { \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`.

```

4228         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4229         { \int_use:N \l_@@_final_i_int }
4230         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4231         { } % for the name of the block
4232     }
4233 }

```

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 known wheter 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.

```

4234 \cs_new_protected:Npn \@@_open_shorten:
4235 {
4236     \bool_if:NT \l_@@_initial_open_bool
4237     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4238     \bool_if:NT \l_@@_final_open_bool
4239     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4240 }

```

The following commmand (*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).

```

4241 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4242 {
4243     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```



```

4244 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4245 \int_set_eq:NN \l_@@_row_max_int \c_iRow
4246 \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`.

```

4247 \seq_if_empty:NF \g_@@_submatrix_seq
4248 {
4249   \seq_map_inline:Nn \g_@@_submatrix_seq
4250   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4251 }
4252 }

```

`#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.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4253 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4254 {
4255   \if_int_compare:w #3 > #1
4256   \else:
4257     \if_int_compare:w #1 > #5
4258     \else:
4259       \if_int_compare:w #4 > #2
4260       \else:
4261         \if_int_compare:w #2 > #6
4262         \else:
4263           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4264           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4265           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4266           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4267         \fi:
4268       \fi:
4269     \fi:
4270   \fi:
4271 }

4272 \cs_new_protected:Npn \@@_set_initial_coords:
4273 {
4274   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4275   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4276 }
4277 \cs_new_protected:Npn \@@_set_final_coords:
4278 {

```

```

4279 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4280 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4281 }
4282 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4283 {
4284   \pgfpointanchor
4285   {
4286     \@@_env:
4287     - \int_use:N \l_@@_initial_i_int
4288     - \int_use:N \l_@@_initial_j_int
4289   }
4290   { #1 }
4291   \@@_set_initial_coords:
4292 }
4293 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4294 {
4295   \pgfpointanchor
4296   {
4297     \@@_env:
4298     - \int_use:N \l_@@_final_i_int
4299     - \int_use:N \l_@@_final_j_int
4300   }
4301   { #1 }
4302   \@@_set_final_coords:
4303 }
4304 \cs_new_protected:Npn \@@_open_x_initial_dim:
4305 {
4306   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4307   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4308   {
4309     \cs_if_exist:cT
4310     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4311     {
4312       \pgfpointanchor
4313       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4314       { west }
4315       \dim_set:Nn \l_@@_x_initial_dim
4316       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4317     }
4318   }

```

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

```

4319 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4320 {
4321   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4322   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4323   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4324 }
4325 }
4326 \cs_new_protected:Npn \@@_open_x_final_dim:
4327 {
4328   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4329   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4330   {
4331     \cs_if_exist:cT
4332     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4333     {
4334       \pgfpointanchor
4335       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4336       { east }
4337       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4338       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4339     }

```

```
4340 }
```

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

```
4341 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4342 {
4343   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4344   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4345   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4346 }
4347 }
```

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.

```
4348 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4349 {
4350   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4351   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4352   {
4353     \@@_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.

```
4354   \group_begin:
4355   \@@_open_shorten:
4356   \int_if_zero:nTF { #1 }
4357   { \color { nicematrix-first-row } }
4358   {
```

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.

```
4359     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4360     { \color { nicematrix-last-row } }
4361   }
4362   \keys_set:nn { nicematrix / xdots } { #3 }
4363   \@@_color:o \l_@@_xdots_color_tl
4364   \@@_actually_draw_Ldots:
4365   \group_end:
4366 }
4367 }
```

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`.

```
4368 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4369 {
4370   \bool_if:NTF \l_@@_initial_open_bool
4371   {
4372     \@@_open_x_initial_dim:
4373     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4374     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4375   }
4376   { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4377 \bool_if:NTF \l_@@_final_open_bool
4378 {
4379   \@@_open_x_final_dim:
4380   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4381   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4382 }
4383 { \@@_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.

```

4384 \bool_lazy_all:nTF
4385 {
4386   \l_@@_initial_open_bool
4387   \l_@@_final_open_bool
4388   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4389 }
4390 {
4391   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4392   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4393 }

```

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 (?).

```

4394 {
4395   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4396   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4397 }
4398 \@@_draw_line:
4399 }

```

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.

```

4400 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4401 {
4402   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4403   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4404   {
4405     \@@_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.

```

4406   \group_begin:
4407   \@@_open_shorten:
4408   \int_if_zero:nTF { #1 }
4409   { \color { nicematrix-first-row } }
4410   {

```

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.

```

4411     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4412     { \color { nicematrix-last-row } }
4413   }
4414   \keys_set:nn { nicematrix / xdots } { #3 }
4415   \@@_color:o \l_@@_xdots_color_tl
4416   \@@_actually_draw_Cdots:
4417 \group_end:
4418 }
4419 }

```

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.

```

4420 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4421 {
4422   \bool_if:NTF \l_@@_initial_open_bool
4423     { \@@_open_x_initial_dim: }
4424     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4425   \bool_if:NTF \l_@@_final_open_bool
4426     { \@@_open_x_final_dim: }
4427     { \@@_set_final_coords_from_anchor:n { mid-west } }
4428   \bool_lazy_and:nnTF
4429     \l_@@_initial_open_bool
4430     \l_@@_final_open_bool
4431   {
4432     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4433     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4434     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4435     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4436     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4437   }
4438   {
4439     \bool_if:NT \l_@@_initial_open_bool
4440       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4441     \bool_if:NT \l_@@_final_open_bool
4442       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4443   }
4444   \@@_draw_line:
4445 }

4446 \cs_new_protected:Npn \@@_open_y_initial_dim:
4447 {
4448   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4449   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4450   {
4451     \cs_if_exist:cT
4452       { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4453     {
4454       \pgfpointanchor
4455         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4456         { north }
4457       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4458         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4459     }
4460   }
4461   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4462   {
4463     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4464     \dim_set:Nn \l_@@_y_initial_dim
4465       {
4466         \fp_to_dim:n
4467           {
4468             \pgf@y
4469             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4470           }
4471       }
4472   }
4473 }

```

```

4474 \cs_new_protected:Npn \@@_open_y_final_dim:
4475 {
4476   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4477   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4478   {
4479     \cs_if_exist:cT
4480     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4481     {
4482       \pgfpointanchor
4483       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4484       { south }
4485       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4486       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4487     }
4488   }
4489   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4490   {
4491     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4492     \dim_set:Nn \l_@@_y_final_dim
4493     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4494   }
4495 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

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

```

4502   \group_begin:
4503   \@@_open_shorten:
4504   \int_if_zero:nTF { #2 }
4505   { \color { nicematrix-first-col } }
4506   {
4507     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4508     { \color { nicematrix-last-col } }
4509   }
4510   \keys_set:nn { nicematrix / xdots } { #3 }
4511   \@@_color:o \l_@@_xdots_color_tl
4512   \@@_actually_draw_Vdots:
4513   \group_end:
4514 }
4515 }

```

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`.

```

4516 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4517 {

```

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

```
4518 \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.

```
4519 {
4520 \@@_open_y_initial_dim:
4521 \@@_open_y_final_dim:
4522 \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4523 {
4524 \@@_qpoint:n { col - 1 }
4525 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4526 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4527 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4528 \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4529 }
4530 {
4531 \bool_lazy_and:nnTF
4532 { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4533 { \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”.

```
4534 {
4535 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4536 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4537 \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4538 \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4539 \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4540 }
```

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

```
4541 {
4542 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4543 \dim_set_eq:NN \l_tmpa_dim \pgf@x
4544 \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4545 \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4546 }
4547 }
4548 }
```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```
4549 {
4550 \bool_set_false:N \l_tmpa_bool
4551 \bool_if:NF \l_@@_initial_open_bool
4552 {
4553 \bool_if:NF \l_@@_final_open_bool
4554 {
4555 \@@_set_initial_coords_from_anchor:n { south-west }
4556 \@@_set_final_coords_from_anchor:n { north-west }
4557 \bool_set:Nn \l_tmpa_bool
4558 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4559 }
4560 }
```

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

```
4561 \bool_if:NTF \l_@@_initial_open_bool
4562 {
4563 \@@_open_y_initial_dim:
4564 \@@_set_final_coords_from_anchor:n { north }
4565 \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4566 }
4567 {
4568 \@@_set_initial_coords_from_anchor:n { south }
4569 \bool_if:NTF \l_@@_final_open_bool
```

```
4570 \@@_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.

```
4571 {
4572   \@@_set_final_coords_from_anchor:n { north }
4573   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4574   {
4575     \dim_set:Nn \l_@@_x_initial_dim
4576     {
4577       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4578       \l_@@_x_initial_dim \l_@@_x_final_dim
4579     }
4580   }
4581 }
4582 }
4583 }
4584 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4585 \@@_draw_line:
4586 }
```

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.

```
4587 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4588 {
4589   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4590   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4591   {
4592     \@@_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.

```
4593   \group_begin:
4594     \@@_open_shorten:
4595     \keys_set:nn { nicematrix / xdots } { #3 }
4596     \@@_color:o \l_@@_xdots_color_tl
4597     \@@_actually_draw_Ddots:
4598   \group_end:
4599 }
4600 }
```

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.`

```
4601 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4602 {
4603   \bool_if:NTF \l_@@_initial_open_bool
4604   {
4605     \@@_open_y_initial_dim:
4606     \@@_open_x_initial_dim:
```



```

4607     }
4608     { \@@_set_initial_coords_from_anchor:n { south-east } }
4609     \bool_if:NTF \l_@@_final_open_bool
4610     {
4611         \@@_open_x_final_dim:
4612         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4613     }
4614     { \@@_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.

```

4615     \bool_if:NT \l_@@_parallelize_diags_bool
4616     {
4617         \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).

```

4618         \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.

```

4619         {
4620             \dim_gset:Nn \g_@@_delta_x_one_dim
4621             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4622             \dim_gset:Nn \g_@@_delta_y_one_dim
4623             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4624         }

```

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`.

```

4625         {
4626             \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4627             {
4628                 \dim_set:Nn \l_@@_y_final_dim
4629                 {
4630                     \l_@@_y_initial_dim +
4631                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4632                     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4633                 }
4634             }
4635         }
4636     }
4637     \@@_draw_line:
4638 }

```

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.

```

4639 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4640 {
4641     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4642     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4643     {
4644         \@@_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.

```

4645     \group_begin:
4646     \@@_open_shorten:
4647     \keys_set:nn { nicematrix / xdots } { #3 }
4648     \@@_color:o \l_@@_xdots_color_tl
4649     \@@_actually_draw_Iddots:
4650     \group_end:
4651 }
4652 }

```

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.`

```

4653 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4654 {
4655   \bool_if:NTF \l_@@_initial_open_bool
4656   {
4657     \@@_open_y_initial_dim:
4658     \@@_open_x_initial_dim:
4659   }
4660   { \@@_set_initial_coords_from_anchor:n { south-west } }
4661   \bool_if:NTF \l_@@_final_open_bool
4662   {
4663     \@@_open_y_final_dim:
4664     \@@_open_x_final_dim:
4665   }
4666   { \@@_set_final_coords_from_anchor:n { north-east } }
4667   \bool_if:NT \l_@@_parallelize_diags_bool
4668   {
4669     \int_gincr:N \g_@@_iddots_int
4670     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4671     {
4672       \dim_gset:Nn \g_@@_delta_x_two_dim
4673       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4674       \dim_gset:Nn \g_@@_delta_y_two_dim
4675       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4676     }
4677     {
4678       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4679       {
4680         \dim_set:Nn \l_@@_y_final_dim
4681         {
4682           \l_@@_y_initial_dim +
4683           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4684           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4685         }
4686       }
4687     }
4688   }
4689   \@@_draw_line:
4690 }

```

## 17 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`

```

4691 \cs_new_protected:Npn \@@_draw_line:
4692 {
4693   \pgfrememberpicturepositiononpagetrue
4694   \pgf@relevantforpicturesizefalse
4695   \bool_lazy_or:nnTF
4696     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4697     \l_@@_dotted_bool
4698     \@@_draw_standard_dotted_line:
4699     \@@_draw_unstandard_dotted_line:
4700 }

```

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.

```

4701 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4702 {
4703   \begin { scope }
4704     \@@_draw_unstandard_dotted_line:o
4705     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4706 }

```

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

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

```

4707 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4708 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4709 {
4710   \@@_draw_unstandard_dotted_line:nooo
4711   { #1 }
4712   \l_@@_xdots_up_tl
4713   \l_@@_xdots_down_tl
4714   \l_@@_xdots_middle_tl
4715 }

```

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

```

4716 \hook_gput_code:nnn { begindocument } { . }
4717 {
4718   \IfPackageLoadedT { tikz }
4719   {
4720     \tikzset
4721     {
4722       @@_node_above / .style = { sloped , above } ,
4723       @@_node_below / .style = { sloped , below } ,
4724       @@_node_middle / .style =
4725       {
4726         sloped ,
4727         inner~sep = \c_@@_innersep_middle_dim
4728       }
4729     }
4730   }
4731 }

```

```

4732 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4733 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4734 {

```

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.

```

4735 \dim_zero_new:N \l_@@_l_dim
4736 \dim_set:Nn \l_@@_l_dim
4737 {
4738 \fp_to_dim:n
4739 {
4740 sqrt
4741 (
4742 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4743 +
4744 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4745 )
4746 }
4747 }

```

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.

```

4748 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4749 {
4750 \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4751 \@@_draw_unstandard_dotted_line_i:
4752 }

```

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

```

4753 \bool_if:NT \l_@@_xdots_h_labels_bool
4754 {
4755 \tikzset
4756 {
4757 @@_node_above / .style = { auto = left } ,
4758 @@_node_below / .style = { auto = right } ,
4759 @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4760 }
4761 }
4762 \tl_if_empty:nF { #4 }
4763 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4764 \draw
4765 [ #1 ]
4766 ( \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).

```

4767 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4768 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4769 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4770 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4771 \end { scope }
4772 }
4773 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4774 {
4775 \dim_set:Nn \l_tmpa_dim
4776 {
4777 \l_@@_x_initial_dim
4778 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4779     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4780   }
4781   \dim_set:Nn \l_tmpb_dim
4782   {
4783     \l_@@_y_initial_dim
4784     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4785     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4786   }
4787   \dim_set:Nn \l_@@_tmpc_dim
4788   {
4789     \l_@@_x_final_dim
4790     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4791     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4792   }
4793   \dim_set:Nn \l_@@_tmpd_dim
4794   {
4795     \l_@@_y_final_dim
4796     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4797     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4798   }
4799   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4800   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4801   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4802   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4803 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4804 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4805 {
4806   \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.

```

4807   \dim_zero_new:N \l_@@_l_dim
4808   \dim_set:Nn \l_@@_l_dim
4809   {
4810     \fp_to_dim:n
4811     {
4812       sqrt
4813       (
4814         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4815         +
4816         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4817       )
4818     }
4819   }

```

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.

```

4820   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4821   {
4822     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4823     \@@_draw_standard_dotted_line_i:
4824   }
4825   \group_end:
4826   \bool_lazy_all:nF
4827   {
4828     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4829     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4830     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4831     }
4832     \l_@@_labels_standard_dotted_line:
4833 }
4834 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4835 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4836 {

```

The number of dots will be  $\l_1\text{tmpa\_int} + 1$ .

```

4837     \int_set:Nn \l_1tmpa_int
4838     {
4839         \dim_ratio:nn
4840         {
4841             \l_@@_l_dim
4842             - \l_@@_xdots_shorten_start_dim
4843             - \l_@@_xdots_shorten_end_dim
4844         }
4845         \l_@@_xdots_inter_dim
4846     }

```

The dimensions  $\l_1\text{tmpa\_dim}$  and  $\l_1\text{tmpb\_dim}$  are the coordinates of the vector between two dots in the dotted line.

```

4847     \dim_set:Nn \l_1tmpa_dim
4848     {
4849         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4850         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4851     }
4852     \dim_set:Nn \l_1tmpb_dim
4853     {
4854         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4855         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4856     }

```

In the loop over the dots, the dimensions  $\l_1\text{@@_x\_initial\_dim}$  and  $\l_1\text{@@_y\_initial\_dim}$  will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4857     \dim_gadd:Nn \l_@@_x_initial_dim
4858     {
4859         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4860         \dim_ratio:nn
4861         {
4862             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4863             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4864         }
4865         { 2 \l_@@_l_dim }
4866     }
4867     \dim_gadd:Nn \l_@@_y_initial_dim
4868     {
4869         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4870         \dim_ratio:nn
4871         {
4872             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4873             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4874         }
4875         { 2 \l_@@_l_dim }
4876     }
4877     \pgf@relevantforpicturesizefalse
4878     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4879     {
4880         \pgfpathcircle
4881         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4882         { \l_@@_xdots_radius_dim }
4883         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4884         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4885     }

```

```

4886 \pgfusepathqfill
4887 }

4888 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4889 {
4890   \pgfscope
4891   \pgftransformshift
4892   {
4893     \pgfpointlineatime { 0.5 }
4894     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4895     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4896   }
4897   \fp_set:Nn \l_tmpa_fp
4898   {
4899     atand
4900     (
4901       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4902       \l_@@_x_final_dim - \l_@@_x_initial_dim
4903     )
4904   }
4905   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4906   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4907   \tl_if_empty:NF \l_@@_xdots_middle_tl
4908   {
4909     \begin { pgfscope }
4910     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4911     \pgfnode
4912     { rectangle }
4913     { center }
4914     {
4915       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4916       {
4917         \c_math_toggle_token
4918         \scriptstyle \l_@@_xdots_middle_tl
4919         \c_math_toggle_token
4920       }
4921     }
4922     { }
4923     {
4924       \pgfsetfillcolor { white }
4925       \pgfusepath { fill }
4926     }
4927     \end { pgfscope }
4928   }
4929   \tl_if_empty:NF \l_@@_xdots_up_tl
4930   {
4931     \pgfnode
4932     { rectangle }
4933     { south }
4934     {
4935       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4936       {
4937         \c_math_toggle_token
4938         \scriptstyle \l_@@_xdots_up_tl
4939         \c_math_toggle_token
4940       }
4941     }
4942     { }
4943     { \pgfusepath { } }
4944   }
4945   \tl_if_empty:NF \l_@@_xdots_down_tl
4946   {
4947     \pgfnode

```

```

4948     { rectangle }
4949     { north }
4950     {
4951       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4952       {
4953         \c_math_toggle_token
4954         \scriptstyle \l_@@_xdots_down_tl
4955         \c_math_toggle_token
4956       }
4957     }
4958     { }
4959     { \pgfusepath { } }
4960   }
4961 \endpgfscope
4962 }

```

## 18 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.

```

4963 \hook_gput_code:nnn { begindocument } { . }
4964 {
4965   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4966   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4967   \cs_new_protected:Npn \@@_Ldots
4968     { \@@_collect_options:n { \@@_Ldots_i } }
4969   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4970     {
4971       \int_if_zero:nTF \c@jCol
4972       { \@@_error:nn { in~first~col } \Ldots }
4973       {
4974         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4975         { \@@_error:nn { in~last~col } \Ldots }
4976         {
4977           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4978           { #1 , down = #2 , up = #3 , middle = #4 }
4979         }
4980       }
4981       \bool_if:NF \l_@@_nullify_dots_bool
4982       { \phantom { \ensuremath { \@@_old_ldots } } }
4983       \bool_gset_true:N \g_@@_empty_cell_bool
4984     }

4985   \cs_new_protected:Npn \@@_Cdots
4986     { \@@_collect_options:n { \@@_Cdots_i } }
4987   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4988     {
4989       \int_if_zero:nTF \c@jCol
4990       { \@@_error:nn { in~first~col } \Cdots }
4991       {
4992         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```



```

4993         { \@@_error:nn { in~last~col } \Cdots }
4994     {
4995         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4996         { #1 , down = #2 , up = #3 , middle = #4 }
4997     }
4998 }
4999 \bool_if:NF \l_@@_nullify_dots_bool
5000 { \phantom { \ensuremath { \@@_old_cdots } } }
5001 \bool_gset_true:N \g_@@_empty_cell_bool
5002 }

5003 \cs_new_protected:Npn \@@_Vdots
5004 { \@@_collect_options:n { \@@_Vdots_i } }
5005 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5006 {
5007     \int_if_zero:nTF \c@iRow
5008     { \@@_error:nn { in~first~row } \Vdots }
5009     {
5010         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5011         { \@@_error:nn { in~last~row } \Vdots }
5012         {
5013             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5014             { #1 , down = #2 , up = #3 , middle = #4 }
5015         }
5016     }
5017     \bool_if:NF \l_@@_nullify_dots_bool
5018     { \phantom { \ensuremath { \@@_old_vdots } } }
5019     \bool_gset_true:N \g_@@_empty_cell_bool
5020 }

5021 \cs_new_protected:Npn \@@_Ddots
5022 { \@@_collect_options:n { \@@_Ddots_i } }
5023 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5024 {
5025     \int_case:nnF \c@iRow
5026     {
5027         0 { \@@_error:nn { in~first~row } \Ddots }
5028         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5029     }
5030     {
5031         \int_case:nnF \c@jCol
5032         {
5033             0 { \@@_error:nn { in~first~col } \Ddots }
5034             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5035         }
5036         {
5037             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5038             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5039             { #1 , down = #2 , up = #3 , middle = #4 }
5040         }
5041     }
5042 }
5043 \bool_if:NF \l_@@_nullify_dots_bool
5044 { \phantom { \ensuremath { \@@_old_ddots } } }
5045 \bool_gset_true:N \g_@@_empty_cell_bool
5046 }

5047 \cs_new_protected:Npn \@@_Iddots
5048 { \@@_collect_options:n { \@@_Iddots_i } }
5049 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5050 {

```

```

5051 \int_case:nnF \c@iRow
5052 {
5053     0 { \@@_error:nn { in~first~row } \Iddots }
5054     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5055 }
5056 {
5057     \int_case:nnF \c@jCol
5058     {
5059         0 { \@@_error:nn { in~first~col } \Iddots }
5060         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5061     }
5062     {
5063         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5064         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5065         { #1 , down = #2 , up = #3 , middle = #4 }
5066     }
5067 }
5068 \bool_if:NF \l_@@_nullify_dots_bool
5069 { \phantom { \ensuremath { \@@_old_iddots } } }
5070 \bool_gset_true:N \g_@@_empty_cell_bool
5071 }
5072 }

```

End of the `\AddToHook`.

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

```

5073 \keys_define:nn { nicematrix / Ddots }
5074 {
5075     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5076     draw-first .default:n = true ,
5077     draw-first .value_forbidden:n = true
5078 }

```

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

```

5079 \cs_new_protected:Npn \@@_Hspace:
5080 {
5081     \bool_gset_true:N \g_@@_empty_cell_bool
5082     \hspace
5083 }

```

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`.

```

5084 \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`.

```

5085 \cs_new:Npn \@@_Hdotsfor:
5086 {
5087     \bool_lazy_and:nnTF
5088     { \int_if_zero_p:n \c@jCol }
5089     { \int_if_zero_p:n \l_@@_first_col_int }
5090     {
5091         \bool_if:NTF \g_@@_after_col_zero_bool
5092         {
5093             \multicolumn { 1 } { c } { }
5094             \@@_Hdotsfor_i
5095         }
5096         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5097     }
5098 }

```

```

5099     \multicolumn { 1 } { c } { }
5100     \@@_Hdotsfor_i
5101   }
5102 }

```

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:`).

```

5103 \hook_gput_code:nnn { begindocument } { . }
5104 {
5105   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5106   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5107   \cs_new_protected:Npn \@@_Hdotsfor_i
5108     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5109   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5110     {
5111       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5112       {
5113         \@@_Hdotsfor:nnnn
5114         { \int_use:N \c@iRow }
5115         { \int_use:N \c@jCol }
5116         { #2 }
5117         {
5118           #1 , #3 ,
5119           down = \exp_not:n { #4 } ,
5120           up = \exp_not:n { #5 } ,
5121           middle = \exp_not:n { #6 }
5122         }
5123       }
5124       \prg_replicate:nn { #2 - 1 }
5125       {
5126         &
5127         \multicolumn { 1 } { c } { }
5128         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5129       }
5130     }
5131 }

```

```

5132 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5133 {
5134   \bool_set_false:N \l_@@_initial_open_bool
5135   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5136   \int_set:Nn \l_@@_initial_i_int { #1 }
5137   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5138   \int_compare:nNnTF { #2 } = \c_one_int
5139   {
5140     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5141     \bool_set_true:N \l_@@_initial_open_bool
5142   }
5143   {
5144     \cs_if_exist:cTF
5145     {
5146       pgf @ sh @ ns @ \@@_env:
5147       - \int_use:N \l_@@_initial_i_int
5148       - \int_eval:n { #2 - 1 }
5149     }
5150     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5151     {

```

```

5152         \int_set:Nn \l_@@_initial_j_int { #2 }
5153         \bool_set_true:N \l_@@_initial_open_bool
5154     }
5155 }
5156 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5157 {
5158     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5159     \bool_set_true:N \l_@@_final_open_bool
5160 }
5161 {
5162     \cs_if_exist:cTF
5163     {
5164         pgf @ sh @ ns @ \@@_env:
5165         - \int_use:N \l_@@_final_i_int
5166         - \int_eval:n { #2 + #3 }
5167     }
5168     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5169     {
5170         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5171         \bool_set_true:N \l_@@_final_open_bool
5172     }
5173 }
5174 \group_begin:
5175 \@@_open_shorten:
5176 \int_if_zero:nTF { #1 }
5177 { \color { nicematrix-first-row } }
5178 {
5179     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5180     { \color { nicematrix-last-row } }
5181 }
5182
5183 \keys_set:nn { nicematrix / xdots } { #4 }
5184 \@@_color:o \l_@@_xdots_color_tl
5185 \@@_actually_draw_Ldots:
5186 \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).

```

5187     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5188     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5189 }

```

```

5190 \hook_gput_code:nnn { begindocument } { . }
5191 {
5192     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5193     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5194     \cs_new_protected:Npn \@@_Vdotsfor:
5195     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5196     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5197     {
5198         \bool_gset_true:N \g_@@_empty_cell_bool
5199         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5200         {
5201             \@@_Vdotsfor:nnnn
5202             { \int_use:N \c@iRow }
5203             { \int_use:N \c@jCol }
5204             { #2 }
5205             {
5206                 #1 , #3 ,
5207                 down = \exp_not:n { #4 } ,
5208                 up = \exp_not:n { #5 } ,

```

```

5209         middle = \exp_not:n { #6 }
5210     }
5211 }
5212 }
5213 }

```

```

5214 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5215 {
5216     \bool_set_false:N \l_@@_initial_open_bool
5217     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5218     \int_set:Nn \l_@@_initial_j_int { #2 }
5219     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5220     \int_compare:nNnTF { #1 } = \c_one_int
5221     {
5222         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5223         \bool_set_true:N \l_@@_initial_open_bool
5224     }
5225     {
5226         \cs_if_exist:cTF
5227         {
5228             pgf @ sh @ ns @ \@@_env:
5229             - \int_eval:n { #1 - 1 }
5230             - \int_use:N \l_@@_initial_j_int
5231         }
5232         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5233         {
5234             \int_set:Nn \l_@@_initial_i_int { #1 }
5235             \bool_set_true:N \l_@@_initial_open_bool
5236         }
5237     }
5238     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5239     {
5240         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5241         \bool_set_true:N \l_@@_final_open_bool
5242     }
5243     {
5244         \cs_if_exist:cTF
5245         {
5246             pgf @ sh @ ns @ \@@_env:
5247             - \int_eval:n { #1 + #3 }
5248             - \int_use:N \l_@@_final_j_int
5249         }
5250         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5251         {
5252             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5253             \bool_set_true:N \l_@@_final_open_bool
5254         }
5255     }
5256 \group_begin:
5257 \@@_open_shorten:
5258 \int_if_zero:nTF { #2 }
5259 { \color { nicematrix-first-col } }
5260 {
5261     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5262     { \color { nicematrix-last-col } }
5263 }
5264 \keys_set:nn { nicematrix / xdots } { #4 }
5265 \@@_color:o \l_@@_xdots_color_tl
5266 \@@_actually_draw_Vdots:
5267 \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:nnn`). This declaration is done by defining a special control sequence (to nil).

```

5268 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5269 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5270 }

```

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

```

5271 \NewDocumentCommand \@@_rotate: { 0 { } }
5272 {
5273   \peek_remove_spaces:n
5274   {
5275     \bool_gset_true:N \g_@@_rotate_bool
5276     \keys_set:nn { nicematrix / rotate } { #1 }
5277   }
5278 }

5279 \keys_define:nn { nicematrix / rotate }
5280 {
5281   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5282   c .value_forbidden:n = true ,
5283   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5284 }

```

## 19 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>

```

5285 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5286 {
5287   \tl_if_empty:nTF { #2 }
5288   { #1 }
5289   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5290 }
5291 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5292 { \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).

```

5293 \hook_gput_code:nnn { begindocument } { . }
5294 {

```

---

<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`.

```

5295 \cs_set_nopar:Npn \l_@@_argspec_tl
5296 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5297 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5298 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5299 {
5300   \group_begin:
5301   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5302   \@@_color:o \l_@@_xdots_color_tl
5303   \use:e
5304   {
5305     \@@_line_i:nn
5306     { \@@_double_int_eval:n #2 - \q_stop }
5307     { \@@_double_int_eval:n #3 - \q_stop }
5308   }
5309   \group_end:
5310 }
5311 }

5312 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5313 {
5314   \bool_set_false:N \l_@@_initial_open_bool
5315   \bool_set_false:N \l_@@_final_open_bool
5316   \bool_lazy_or:nnTF
5317   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5318   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5319   { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

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

```

5320 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5321 }

5322 \hook_gput_code:nnn { begindocument } { . }
5323 {
5324   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5325   {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:.`

```

5326 \c_@@_pgfortikzpicture_tl
5327 \@@_draw_line_iii:nn { #1 } { #2 }
5328 \c_@@_endpgfortikzpicture_tl
5329 }
5330 }

```

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

```

5331 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5332 {
5333   \pgfrememberpicturepositiononpagetrue
5334   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5335   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5336   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5337   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5338   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5339   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5340   \@@_draw_line:
5341 }

```

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).

## 20 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

```
5342 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5343 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

5344 \cs_new:Npn \@@_if_col_greater_than:nn #1 #2
5345 { \int_compare:nNnF { \c@jCol } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5346 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5347 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5348 {
5349   \tl_gput_right:Ne \g_@@_row_style_tl
5350   {
```

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

```
5351   \exp_not:N
5352   \@@_if_row_less_than:nn
5353   { \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`).

```
5354   {
5355     \exp_not:N
5356     \@@_if_col_greater_than:nn
5357     { \int_eval:n { \c@jCol } }
5358     { \exp_not:n { #1 } \scan_stop: }
5359   }
5360 }
5361 }
```

```
5362 \keys_define:nn { nicematrix / RowStyle }
5363 {
5364   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5365   cell-space-top-limit .value_required:n = true ,
5366   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5367   cell-space-bottom-limit .value_required:n = true ,
5368   cell-space-limits .meta:n =
5369   {
5370     cell-space-top-limit = #1 ,
5371     cell-space-bottom-limit = #1 ,
5372   } ,
5373   color .tl_set:N = \l_@@_color_tl ,
5374   color .value_required:n = true ,
5375   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5376   bold .default:n = true ,
5377   nb-rows .code:n =
5378   \str_if_eq:eeTF { #1 } { * }
5379   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5380   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
```



```

5381     nb-rows .value_required:n = true ,
5382     fill .tl_set:N = \l_@@_fill_tl ,
5383     fill .value_required:n = true ,
5384     opacity .tl_set:N = \l_@@_opacity_tl ,
5385     opacity .value_required:n = true ,
5386     rowcolor .tl_set:N = \l_@@_fill_tl ,
5387     rowcolor .value_required:n = true ,
5388     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5389     rounded-corners .default:n = 4 pt ,
5390     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5391 }

```

```

5392 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5393 {
5394   \group_begin:
5395   \tl_clear:N \l_@@_fill_tl
5396   \tl_clear:N \l_@@_opacity_tl
5397   \tl_clear:N \l_@@_color_tl
5398   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5399   \dim_zero:N \l_@@_rounded_corners_dim
5400   \dim_zero:N \l_tmpa_dim
5401   \dim_zero:N \l_tmpb_dim
5402   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key rowcolor (of its alias fill) has been used.

```

5403   \tl_if_empty:NF \l_@@_fill_tl
5404   {
5405     \@@_add_opacity_to_fill:
5406     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5407     {

```

First, the case when the command \RowStyle is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command \RowStyle is issued, but in the other whole rows, if the key nb-rows is used.

```

5408       \int_compare:nNnTF \c@jCol > \c_one_int
5409       {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row). The command \@@\_exp\_color\_arg:No is *fully expandable*.

```

5410         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5411         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5412         { \int_use:N \c@iRow - * }
5413         { \dim_use:N \l_@@_rounded_corners_dim }

```

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

```

5414         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5415         { \@@_rounded_from_row:n { \c@iRow + 1 } }
5416     }

```

Now, directly all the rows in the case of a command \RowStyle issued in the first column of the array.

```

5417     { \@@_rounded_from_row:n { \c@iRow } }
5418   }
5419 }
5420 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l\_tmpa\_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5421   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5422   {
5423     \@@_put_in_row_style:e
5424     {
5425       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5426       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5427         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5428         { \dim_use:N \l_tmpa_dim }
5429     }
5430 }
5431 }

```

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

```

5432 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5433 {
5434     \@@_put_in_row_style:e
5435     {
5436         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5437         {
5438             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5439             { \dim_use:N \l_tmpb_dim }
5440         }
5441     }
5442 }

```

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

```

5443 \tl_if_empty:NF \l_@@_color_tl
5444 {
5445     \@@_put_in_row_style:e
5446     {
5447         \mode_leave_vertical:
5448         \@@_color:n { \l_@@_color_tl }
5449     }
5450 }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5451 \bool_if:NT \l_@@_bold_row_style_bool
5452 {
5453     \@@_put_in_row_style:n
5454     {
5455         \exp_not:n
5456         {
5457             \if_mode_math:
5458                 \c_math_toggle_token
5459                 \bfseries \boldmath
5460                 \c_math_toggle_token
5461             \else:
5462                 \bfseries \boldmath
5463             \fi:
5464         }
5465     }
5466 }
5467 \group_end:
5468 \g_@@_row_style_tl
5469 \ignorespaces
5470 }

```

The following commande must *not* be protected.

```

5471 \cs_new:Npn \@@_rounded_from_row:n #1
5472 {
5473     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5474     { \int_eval:n { #1 } - 1 }
5475     {
5476         \int_eval:n { \c_iRow + \l_@@_key_nb_rows_int - 1 }
5477         - \exp_not:n { \int_use:N \c_jCol }
5478     }
5479     { \dim_use:N \l_@@_rounded_corners_dim }
5480 }

```

## 21 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).

```
5481 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5482 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5483 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5484 {
```

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.

```
5485 \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`. `\str_if_in:nnF` is mandatory: don't use `\tl_if_in:nnF`.

```
5486 \str_if_in:nnF { #1 } { !! }
5487 {
5488 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5489 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5490 }
5491 \int_if_zero:nTF \l_tmpa_int
```

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

```
5492 {
5493 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5494 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5495 }
```

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

```
5496 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5497 }
```

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

```
5498 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5499 {
5500 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5501 {
```

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

```

5502     \group_begin:
5503     \pgfsetcornersarced
5504     {
5505         \pgfpoint
5506         { \l_@@_tab_rounded_corners_dim }
5507         { \l_@@_tab_rounded_corners_dim }
5508     }

```

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.

```

5509     \bool_if:NTF \l_@@_hvlines_bool
5510     {
5511         \pgfpathrectanglecorners
5512         {
5513             \pgfpointadd
5514             { \@@_qpoint:n { row-1 } }
5515             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5516         }
5517         {
5518             \pgfpointadd
5519             {
5520                 \@@_qpoint:n
5521                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5522             }
5523             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5524         }
5525     }
5526     {
5527         \pgfpathrectanglecorners
5528         { \@@_qpoint:n { row-1 } }
5529         {
5530             \pgfpointadd
5531             {
5532                 \@@_qpoint:n
5533                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5534             }
5535             { \pgfpoint \c_zero_dim \arrayrulewidth }
5536         }
5537     }
5538     \pgfusepath { clip }
5539     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5540     }
5541 }

```

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`).

```

5542 \cs_new_protected:Npn \@@_actually_color:
5543 {
5544     \pgfpicture
5545     \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.

```

5546     \@@_clip_with_rounded_corners:
5547     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5548     {
5549         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5550     {
5551         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5552         \use:c { g_@@_color _ 1 _tl }
5553         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5554     }
5555     {
5556         \begin { pgfscope }
5557             \@@_color_opacity ##2
5558             \use:c { g_@@_color _ ##1 _tl }
5559             \tl_gclear:c { g_@@_color _ ##1 _tl }
5560             \pgfusepath { fill }
5561         \end { pgfscope }
5562     }
5563 }
5564 \endpgfpicture
5565 }

```

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

```

5566 \cs_new_protected:Npn \@@_color_opacity
5567 {
5568     \peek_meaning:NTF [
5569         { \@@_color_opacity:w }
5570         { \@@_color_opacity:w [ ] }
5571     }

```

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.

```

5572 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5573 {
5574     \tl_clear:N \l_tmpa_tl
5575     \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.

```

5576     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5577     \tl_if_empty:NTF \l_tmpb_tl
5578         { \@declaredcolor }
5579         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5580 }

```

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

```

5581 \keys_define:nn { nicematrix / color-opacity }
5582 {
5583     opacity .tl_set:N          = \l_tmpa_tl ,
5584     opacity .value_required:n = true
5585 }

5586 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5587 {
5588     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5589     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5590     \@@_cartesian_path:
5591 }

```

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

```

5592 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5593 {
5594     \tl_if_blank:nF { #2 }
5595     {

```

```

5596 \@@_add_to_colors_seq:en
5597 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5598 { \@@_cartesian_color:nn { #3 } { - } }
5599 }
5600 }

```

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

```

5601 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5602 {
5603   \tl_if_blank:nF { #2 }
5604   {
5605     \@@_add_to_colors_seq:en
5606     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5607     { \@@_cartesian_color:nn { - } { #3 } }
5608   }
5609 }

```

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

```

5610 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5611 {
5612   \tl_if_blank:nF { #2 }
5613   {
5614     \@@_add_to_colors_seq:en
5615     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5616     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5617   }
5618 }

```

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

```

5619 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5620 {
5621   \tl_if_blank:nF { #2 }
5622   {
5623     \@@_add_to_colors_seq:en
5624     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5625     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5626   }
5627 }

```

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

```

5628 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5629 {
5630   \@@_cut_on_hyphen:w #1 \q_stop
5631   \tl_clear_new:N \l_@@_tmpc_tl
5632   \tl_clear_new:N \l_@@_tmpd_tl
5633   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5634   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5635   \@@_cut_on_hyphen:w #2 \q_stop
5636   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5637   \tl_set:Ne \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`.

```

5638 \@@_cartesian_path:n { #3 }
5639 }

```

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

```

5640 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5641 {
5642   \clist_map_inline:nn { #3 }
5643   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5644 }

```

```

5645 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5646 {
5647   \int_step_inline:nn \c@iRow
5648   {
5649     \int_step_inline:nn \c@jCol
5650     {
5651       \int_if_even:nTF { ####1 + ##1 }
5652       { \@@_cellcolor [ #1 ] { #2 } }
5653       { \@@_cellcolor [ #1 ] { #3 } }
5654       { ##1 - ####1 }
5655     }
5656   }
5657 }

```

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”.

```

5658 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5659 {
5660   \@@_rectanglecolor [ #1 ] { #2 }
5661   { 1 - 1 }
5662   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5663 }

```

```

5664 \keys_define:nn { nicematrix / rowcolors }
5665 {
5666   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5667   respect-blocks .default:n = true ,
5668   cols .tl_set:N = \l_@@_cols_tl ,
5669   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5670   restart .default:n = true ,
5671   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5672 }

```

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*.

```

5673 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5674 {

```

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

```

5675   \group_begin:
5676   \seq_clear_new:N \l_@@_colors_seq
5677   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5678   \tl_clear_new:N \l_@@_cols_tl
5679   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5680   \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).

```

5681   \int_zero_new:N \l_@@_color_int
5682   \int_set_eq:NN \l_@@_color_int \c_one_int
5683   \bool_if:NT \l_@@_respect_blocks_bool
5684   {

```

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 the sequence `\l_tmpa_seq`).

```

5685     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5686     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5687     { \@@_not_in_exterior_p:nnnnn ##1 }
5688   }
5689   \pgfpicture
5690   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5691   \clist_map_inline:nn { #2 }
5692   {
5693     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5694     \tl_if_in:NnTF \l_tmpa_tl { - }
5695     { \@@_cut_on_hyphen:w ##1 \q_stop }
5696     { \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.

```

5697     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5698     \int_set:Nn \l_@@_color_int
5699     { \bool_if:NnTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5700     \int_zero_new:N \l_@@_tmpc_int
5701     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5702     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5703     {

```

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

```

5704         \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).

```

5705         \bool_if:NT \l_@@_respect_blocks_bool
5706         {
5707           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5708           { \@@_intersect_our_row_p:nnnnn #####1 }
5709           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

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

```

5710         }
5711         \tl_set:No \l_@@_rows_tl
5712         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5713         \tl_clear_new:N \l_@@_color_tl
5714         \tl_set:Ne \l_@@_color_tl
5715         {
5716           \@@_color_index:n
5717           {
5718             \int_mod:nn
5719             { \l_@@_color_int - 1 }
5720             { \seq_count:N \l_@@_colors_seq }
5721             + 1
5722           }
5723         }
5724         \tl_if_empty:NF \l_@@_color_tl
5725         {
5726           \@@_add_to_colors_seq:ee
5727           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5728           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5729         }
5730         \int_incr:N \l_@@_color_int
5731         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5732       }
5733     }
5734   \endpgfpicture

```



```

5735 \group_end:
5736 }

```

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.

```

5737 \cs_new:Npn \@@_color_index:n #1
5738 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5739 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5740 { \@@_color_index:n { #1 - 1 } }
5741 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5742 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5743 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5744 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5745 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5746 {
5747 \int_compare:nNnT { #3 } > \l_tmpb_int
5748 { \int_set:Nn \l_tmpb_int { #3 } }
5749 }

```

```

5750 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5751 {
5752 \int_if_zero:nTF { #4 }
5753 \prg_return_false:
5754 {
5755 \int_compare:nNnTF { #2 } > \c_jCol
5756 \prg_return_false:
5757 \prg_return_true:
5758 }
5759 }

```

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

```

5760 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5761 {
5762 \int_compare:nNnTF { #1 } > \l_tmpa_int
5763 \prg_return_false:
5764 {
5765 \int_compare:nNnTF \l_tmpa_int > { #3 }
5766 \prg_return_false:
5767 \prg_return_true:
5768 }
5769 }

```

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`).

```

5770 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5771 {
5772 \dim_compare:nNnTF { #1 } = \c_zero_dim

```

```

5773 {
5774   \bool_if:NTF
5775     \l_@@_nocolor_used_bool
5776     \@@_cartesian_path_normal_ii:
5777   {
5778     \clist_if_empty:NTF \l_@@_corners_cells_clist
5779       { \@@_cartesian_path_normal_i:n { #1 } }
5780       \@@_cartesian_path_normal_ii:
5781   }
5782 }
5783 { \@@_cartesian_path_normal_i:n { #1 } }
5784 }

```

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.

```

5785 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5786 {
5787   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5788   \clist_map_inline:Nn \l_@@_cols_tl
5789   {
5790     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5791     \tl_if_in:NnTF \l_tmpa_tl { - }
5792       { \@@_cut_on_hyphen:w ##1 \q_stop }
5793       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5794     \tl_if_empty:NTF \l_tmpa_tl
5795       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5796       {
5797         \str_if_eq:eeT \l_tmpa_tl { * }
5798         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5799       }
5800     \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5801       { \@@_error:n { Invalid~col~number } }
5802     \tl_if_empty:NTF \l_tmpb_tl
5803       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5804       {
5805         \str_if_eq:eeT \l_tmpb_tl { * }
5806         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5807       }
5808     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5809       { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

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

```

5810     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5811     \@@_qpoint:n { col - \l_tmpa_tl }
5812     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5813       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5814       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5815     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5816     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5817   \clist_map_inline:Nn \l_@@_rows_tl
5818   {
5819     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5820     \tl_if_in:NnTF \l_tmpa_tl { - }
5821       { \@@_cut_on_hyphen:w #####1 \q_stop }
5822       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5823     \tl_if_empty:NTF \l_tmpa_tl
5824       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5825       {
5826         \str_if_eq:eeT \l_tmpa_tl { * }
5827         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }

```

```

5828     }
5829     \tl_if_empty:NTF \l_tmpb_tl
5830     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5831     {
5832         \str_if_eq:eeT \l_tmpb_tl { * }
5833         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5834     }
5835     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5836     { \@@_error:n { Invalid~row~number } }
5837     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5838     { \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.

```

5839     \cs_if_exist:cF
5840     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5841     {
5842         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5843         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5844         \@@_qpoint:n { row - \l_tmpa_tl }
5845         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5846         \pgfpathrectanglecorners
5847         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5848         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5849     }
5850 }
5851 }
5852 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key **corners** is used).

```

5853 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5854 {
5855     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5856     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5857     \clist_map_inline:Nn \l_@@_cols_tl
5858     {
5859         \@@_qpoint:n { col - ##1 }
5860         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5861         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5862         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5863         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5864         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5865     \clist_map_inline:Nn \l_@@_rows_tl
5866     {
5867         \@@_if_in_corner:nF { #####1 - ##1 }
5868         {
5869             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5870             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5871             \@@_qpoint:n { row - #####1 }
5872             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5873             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5874             {
5875                 \pgfpathrectanglecorners
5876                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5877                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5878             }
5879         }
5880     }
5881 }
5882 }

```

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`).

```
5883 \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

```
5884 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5885 {
5886   \bool_set_true:N \l_@@_nocolor_used_bool
5887   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5888   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5889   \clist_map_inline:Nn \l_@@_rows_tl
5890   {
5891     \clist_map_inline:Nn \l_@@_cols_tl
5892     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5893   }
5894 }
```

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`.

```
5895 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5896 {
5897   \clist_set_eq:NN \l_tmpa_clist #1
5898   \clist_clear:N #1
5899   \clist_map_inline:Nn \l_tmpa_clist
5900   {
5901     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5902     \tl_if_in:NnTF \l_tmpa_tl { - }
5903     { \@@_cut_on_hyphen:w ##1 \q_stop }
5904     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5905     \bool_lazy_or:nnT
5906     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5907     { \tl_if_blank_p:o \l_tmpa_tl }
5908     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5909     \bool_lazy_or:nnT
5910     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5911     { \tl_if_blank_p:o \l_tmpb_tl }
5912     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5913     \int_compare:nNnT \l_tmpb_tl > #2
5914     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5915     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5916     { \clist_put_right:Nn #1 { ####1 } }
5917   }
5918 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5919 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5920 {
5921   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5922   {
```

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`).

```
5923     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5924     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5925   }
5926   \ignorespaces
5927 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

5928 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5929 {
5930   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5931   {
5932     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5933     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5934     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5935   }
5936   \ignorespaces
5937 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5938 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5939 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5940 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5941 {
5942   \peek_remove_spaces:n
5943   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5944 }

```

```

5945 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5946 {

```

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`.

```

5947   \seq_gclear:N \g_tmpa_seq
5948   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5949   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5950   \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`.

```

5951   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5952   {
5953     { \int_use:N \c@iRow }
5954     { \exp_not:n { #1 } }
5955     { \exp_not:n { #2 } }
5956     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5957   }
5958 }

```

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`).

```

5959 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5960 {
5961   \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).

```

5962     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5963     {
5964       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5965       {
5966         \@@_rowlistcolors
5967         [ \exp_not:n { #2 } ]
5968         { #1 - \int_eval:n { \c@iRow - 1 } }
5969         { \exp_not:n { #3 } }
5970         [ \exp_not:n { #4 } ]
5971       }
5972     }
5973 }

```

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.

```

5974 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5975 {
5976   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5977   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5978   \seq_gclear:N \g_@@_rowlistcolors_seq
5979 }

5980 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5981 {
5982   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5983   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5984 }

```

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.

```

5985 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5986 {

```

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).

```

5987   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5988   {

```

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).

```

5989     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5990     {
5991       \exp_not:N \columncolor [ #1 ]
5992       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5993     }
5994   }
5995 }

5996 \hook_gput_code:nnn { begindocument } { . }
5997 {
5998   \IfPackageLoadedTF { colortbl }
5999   {
6000     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6001     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6002     \cs_new_protected:Npn \@@_revert_colortbl:

```

```

6003         {
6004             \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6005             {
6006                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6007                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6008             }
6009         }
6010     }
6011     { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6012 }

6013 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6014 {
6015     \clist_map_inline:nn { #1 }
6016     {
6017         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6018         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6019         \columncolor { nocolor } { ##1 }
6020     }
6021 }

6022 \cs_new_protected:Npn \@@_EmptyRow:n #1
6023 {
6024     \clist_map_inline:nn { #1 }
6025     {
6026         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6027         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6028         \rowcolor { nocolor } { ##1 }
6029     }
6030 }

```

## 22 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`.

```

6031 \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`.

```

6032 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6033 {
6034     \int_if_zero:nTF \l_@@_first_col_int
6035     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6036     {
6037         \int_if_zero:nTF \c@jCol
6038         {
6039             \int_compare:nNf \c@iRow = { -1 }
6040             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6041         }
6042         { \@@_OnlyMainNiceMatrix_i:n { #1 } }

```

```

6043     }
6044 }

```

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.

```

6045 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6046 {
6047   \int_if_zero:nF \c@iRow
6048   {
6049     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6050     {
6051       \int_compare:nNnT \c@jCol > \c_zero_int
6052       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6053     }
6054   }
6055 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to  $-2$  or  $-1$  (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6056 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6057 {
6058   \IfPackageLoadedTF { tikz }
6059   {
6060     \IfPackageLoadedTF { booktabs }
6061     { #2 }
6062     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6063   }
6064   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6065 }
6066 \NewExpandableDocumentCommand { \@@_TopRule } { }
6067 { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }
6068 \cs_new:Npn \@@_TopRule_i:
6069 {
6070   \noalign \bgroup
6071   \peek_meaning:NTF [
6072   { \@@_TopRule_ii: }
6073   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6074 }
6075 \NewDocumentCommand \@@_TopRule_ii: { o }
6076 {
6077   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6078   {
6079     \@@_hline:n
6080     {
6081       position = \int_eval:n { \c@iRow + 1 } ,
6082       tikz =
6083       {
6084         line-width = #1 ,
6085         yshift = 0.25 \arrayrulewidth ,
6086         shorten< = - 0.5 \arrayrulewidth
6087       } ,
6088       total-width = #1
6089     }
6090   }
6091   \skip_vertical:n { \belowrulesep + #1 }
6092   \egroup
6093 }

```



```

6094 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6095 { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }
6096 \cs_new:Npn \@@_BottomRule_i:
6097 {
6098   \noalign \bgroup
6099   \peek_meaning:NTF [
6100     { \@@_BottomRule_ii: }
6101     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6102   }
6103 \NewDocumentCommand \@@_BottomRule_ii: { o }
6104 {
6105   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6106   {
6107     \@@_hline:n
6108     {
6109       position = \int_eval:n { \c@iRow + 1 } ,
6110       tikz =
6111       {
6112         line-width = #1 ,
6113         yshift = 0.25 \arrayrulewidth ,
6114         shorten-< = - 0.5 \arrayrulewidth
6115       } ,
6116       total-width = #1 ,
6117     }
6118   }
6119   \skip_vertical:N \aboverulesep
6120   \@@_create_row_node_i:
6121   \skip_vertical:n { #1 }
6122   \egroup
6123 }
6124 \NewExpandableDocumentCommand { \@@_MidRule } { }
6125 { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }
6126 \cs_new:Npn \@@_MidRule_i:
6127 {
6128   \noalign \bgroup
6129   \peek_meaning:NTF [
6130     { \@@_MidRule_ii: }
6131     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6132   }
6133 \NewDocumentCommand \@@_MidRule_ii: { o }
6134 {
6135   \skip_vertical:N \aboverulesep
6136   \@@_create_row_node_i:
6137   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6138   {
6139     \@@_hline:n
6140     {
6141       position = \int_eval:n { \c@iRow + 1 } ,
6142       tikz =
6143       {
6144         line-width = #1 ,
6145         yshift = 0.25 \arrayrulewidth ,
6146         shorten-< = - 0.5 \arrayrulewidth
6147       } ,
6148       total-width = #1 ,
6149     }
6150   }
6151   \skip_vertical:n { \belowrulesep + #1 }
6152   \egroup
6153 }

```

## 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.

```

6154 \keys_define:nn { nicematrix / Rules }
6155 {
6156   position .int_set:N = \l_@@_position_int ,
6157   position .value_required:n = true ,
6158   start .int_set:N = \l_@@_start_int ,
6159   end .code:n =
6160     \bool_lazy_or:nnTF
6161     { \tl_if_empty_p:n { #1 } }
6162     { \str_if_eq_p:ee { #1 } { last } }
6163     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6164     { \int_set:Nn \l_@@_end_int { #1 } }
6165 }

```

It’s possible that the rule won’t be drawn continuously from `start` ot `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

6166 \keys_define:nn { nicematrix / RulesBis }
6167 {
6168   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6169   multiplicity .initial:n = 1 ,
6170   dotted .bool_set:N = \l_@@_dotted_bool ,
6171   dotted .initial:n = false ,
6172   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]`.

```

6173   color .code:n =
6174     \@@_set_CT@arc@:n { #1 }
6175     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6176   color .value_required:n = true ,
6177   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6178   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.

```

6179   tikz .code:n =
6180     \IfPackageLoadedTF { tikz }
6181     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6182     { \@@_error:n { tikz-without-tikz } } ,
6183   tikz .value_required:n = true ,
6184   total-width .dim_set:N = \l_@@_rule_width_dim ,
6185   total-width .value_required:n = true ,
6186   width .meta:n = { total-width = #1 } ,
6187   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6188 }

```

## The vertical rules

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

```

6189 \cs_new_protected:Npn \@@_vline:n #1
6190 {

```

The group is for the options.

```

6191 \group_begin:
6192 \int_set_eq:NN \l_@@_end_int \c@iRow
6193 \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).

```

6194 \int_compare:nNt \l_@@_position_int < { \c@jCol + 2 }
6195 \@@_vline_i:
6196 \group_end:
6197 }

6198 \cs_new_protected:Npn \@@_vline_i:
6199 {

```

`\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`.

```

6200 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6201 \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6202 \l_tmpa_tl
6203 {

```

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.

```

6204 \bool_gset_true:N \g_tmpa_bool

6205 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6206 { \@@_test_vline_in_block:nnnnn ##1 }
6207 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6208 { \@@_test_vline_in_block:nnnnn ##1 }
6209 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6210 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6211 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6212 \bool_if:NTF \g_tmpa_bool
6213 {
6214 \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.

```

6215 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6216 }
6217 {
6218 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6219 {
6220 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6221 \@@_vline_ii:
6222 \int_zero:N \l_@@_local_start_int
6223 }
6224 }
6225 }
6226 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6227 {
6228 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6229 \@@_vline_ii:
6230 }
6231 }

```

```

6232 \cs_new_protected:Npn \@@_test_in_corner_v:
6233 {
6234 \int_compare:nNtTF \l_tmpb_tl = { \c@jCol + 1 }
6235 {
6236 \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }

```

```

6237         { \bool_set_false:N \g_tmpa_bool }
6238     }
6239     {
6240         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6241         {
6242             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6243             { \bool_set_false:N \g_tmpa_bool }
6244             {
6245                 \@@_if_in_corner:nT
6246                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6247                 { \bool_set_false:N \g_tmpa_bool }
6248             }
6249         }
6250     }
6251 }

6252 \cs_new_protected:Npn \@@_vline_ii:
6253 {
6254     \tl_clear:N \l_@@_tikz_rule_tl
6255     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6256     \bool_if:NTF \l_@@_dotted_bool
6257     \@@_vline_iv:
6258     {
6259         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6260         \@@_vline_iii:
6261         \@@_vline_v:
6262     }
6263 }

```

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

```

6264 \cs_new_protected:Npn \@@_vline_iii:
6265 {
6266     \pgfpicture
6267     \pgfrememberpicturepositiononpagetrue
6268     \pgf@relevantforpicturesizefalse
6269     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6270     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6271     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6272     \dim_set:Nn \l_tmpb_dim
6273     {
6274         \pgf@x
6275         - 0.5 \l_@@_rule_width_dim
6276         +
6277         ( \arrayrulewidth * \l_@@_multiplicity_int
6278           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6279     }
6280     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6281     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6282     \bool_lazy_all:nT
6283     {
6284         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6285         { \cs_if_exist_p:N \CT@drsc@ }
6286         { ! \tl_if_blank_p:o \CT@drsc@ }
6287     }
6288     {
6289         \group_begin:
6290         \CT@drsc@
6291         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6292         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6293         \dim_set:Nn \l_@@_tmpd_dim
6294         {
6295             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )

```

```

6296         * ( \l_@@_multiplicity_int - 1 )
6297     }
6298     \pgfpathrectanglecorners
6299     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6300     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6301     \pgfusepath { fill }
6302     \group_end:
6303 }
6304 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6305 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6306 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6307 {
6308     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6309     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6310     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6311     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6312 }
6313 \CT@arc@
6314 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6315 \pgfsetrectcap
6316 \pgfusepathqstroke
6317 \endpgfpicture
6318 }

```

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

```

6319 \cs_new_protected:Npn \@@_vline_iv:
6320 {
6321     \pgfpicture
6322     \pgfrememberpicturepositiononpagetrue
6323     \pgf@relevantforpicturesizefalse
6324     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6325     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6326     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6327     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6328     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6329     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6330     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6331     \CT@arc@
6332     \@@_draw_line:
6333     \endpgfpicture
6334 }

```

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

```

6335 \cs_new_protected:Npn \@@_vline_v:
6336 {
6337     \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).

```

6338     \CT@arc@
6339     \tl_if_empty:NF \l_@@_rule_color_tl
6340     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6341     \pgfrememberpicturepositiononpagetrue
6342     \pgf@relevantforpicturesizefalse
6343     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6344     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6345     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6346     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6347     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6348     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6349     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6350     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }

```

```

6351      ( \l_tmpb_dim , \l_tmpa_dim ) --
6352      ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6353 \end { tikzpicture }
6354 }

```

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).

```

6355 \cs_new_protected:Npn \@@_draw_vlines:
6356 {
6357   \int_step_inline:nnn
6358     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6359     {
6360       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6361       \c@jCol
6362       { \int_eval:n { \c@jCol + 1 } }
6363     }
6364     {
6365       \str_if_eq:eeF \l_@@_vlines_clist { all }
6366       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6367       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6368     }
6369 }

```

## 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}`.

```

6370 \cs_new_protected:Npn \@@_hline:n #1
6371 {

```

The group is for the options.

```

6372   \group_begin:
6373   \int_zero_new:N \l_@@_end_int
6374   \int_set_eq:NN \l_@@_end_int \c@jCol
6375   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6376   \@@_hline_i:
6377   \group_end:
6378 }
6379 \cs_new_protected:Npn \@@_hline_i:
6380 {
6381   \int_zero_new:N \l_@@_local_start_int
6382   \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`.

```

6383   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6384   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6385     \l_tmpb_tl
6386   {

```

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.

```

6387     \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6388     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6389     { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6390 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6391 { \@@_test_hline_in_block:nnnnn ##1 }
6392 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6393 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6394 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6395 \bool_if:NTF \g_tmpa_bool
6396 {
6397   \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.

```

6398     { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6399   }
6400   {
6401     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6402     {
6403       \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6404       \@@_hline_ii:
6405       \int_zero:N \l_@@_local_start_int
6406     }
6407   }
6408 }
6409 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6410 {
6411   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6412   \@@_hline_ii:
6413 }
6414 }

6415 \cs_new_protected:Npn \@@_test_in_corner_h:
6416 {
6417   \int_compare:nNnTF \l_tmpa_tl = { \c_iRow + 1 }
6418   {
6419     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6420     { \bool_set_false:N \g_tmpa_bool }
6421   }
6422   {
6423     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6424     {
6425       \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6426       { \bool_set_false:N \g_tmpa_bool }
6427       {
6428         \@@_if_in_corner:nT
6429         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6430         { \bool_set_false:N \g_tmpa_bool }
6431       }
6432     }
6433   }
6434 }

6435 \cs_new_protected:Npn \@@_hline_ii:
6436 {
6437   \tl_clear:N \l_@@_tikz_rule_tl
6438   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6439   \bool_if:NTF \l_@@_dotted_bool
6440   \@@_hline_iv:
6441   {
6442     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6443     \@@_hline_iii:
6444     \@@_hline_v:
6445   }
6446 }

```

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

```

6447 \cs_new_protected:Npn \@@_hline_iii:
6448 {
6449   \pgfpicture
6450   \pgfrememberpicturepositiononpagetrue
6451   \pgf@relevantforpicturesizefalse
6452   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6453   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6454   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6455   \dim_set:Nn \l_tmpb_dim
6456     {
6457       \pgf@y
6458       - 0.5 \l_@@_rule_width_dim
6459       +
6460       ( \arrayrulewidth * \l_@@_multiplicity_int
6461         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6462     }
6463   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6464   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6465   \bool_lazy_all:nT
6466     {
6467       { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6468       { \cs_if_exist_p:N \CT@drsc@ }
6469       { ! \tl_if_blank_p:o \CT@drsc@ }
6470     }
6471     {
6472       \group_begin:
6473       \CT@drsc@
6474       \dim_set:Nn \l_@@_tmpd_dim
6475         {
6476           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6477           * ( \l_@@_multiplicity_int - 1 )
6478         }
6479       \pgfpathrectanglecorners
6480         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6481         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6482       \pgfusepathqfill
6483       \group_end:
6484     }
6485   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6486   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6487   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6488     {
6489       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6490       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6491       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6492       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6493     }
6494   \CT@arc@
6495   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6496   \pgfsetrectcap
6497   \pgfusepathqstroke
6498   \endpgfpicture
6499 }

```

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



```

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

```

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

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

```

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

```

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

```

6500 \cs_new_protected:Npn \@@_hline_iv:
6501 {
6502   \pgfpicture
6503   \pgfrememberpicturepositiononpagetrue
6504   \pgf@relevantforpicturesizefalse
6505   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6506   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6507   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6508   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6509   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6510   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6511   {
6512     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6513     \bool_if:NF \g_@@_delims_bool
6514     { \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.

```

6515     \tl_if_eq:NnF \g_@@_left_delim_tl (
6516       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6517     )
6518     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6519     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6520     \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6521     {
6522       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6523       \bool_if:NF \g_@@_delims_bool
6524       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6525       \tl_if_eq:NnF \g_@@_right_delim_tl (
6526         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6527       )
6528     }
6529     \CT@arc@
6530     \@@_draw_line:
6531     \endpgfpicture

```

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`).

```

6532 \cs_new_protected:Npn \@@_hline_v:
6533 {
6534   \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).

```

6535     \CT@arc@
6536     \tl_if_empty:NF \l_@@_rule_color_tl

```

```

6537     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6538     \pgfrememberpicturepositiononpagetrue
6539     \pgf@relevantforpicturesizefalse
6540     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6541     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6542     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6543     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6544     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6545     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6546     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6547     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6548     ( \l_tmpa_dim , \l_tmpb_dim ) --
6549     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6550     \end { tikzpicture }
6551 }

```

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).

```

6552 \cs_new_protected:Npn \@@_draw_hlines:
6553 {
6554   \int_step_inline:nnn
6555   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6556   {
6557     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6558     \c@iRow
6559     { \int_eval:n { \c@iRow + 1 } }
6560   }
6561   {
6562     \str_if_eq:eeF \l_@@_hlines_clist { all }
6563     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6564     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6565   }
6566 }

```

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

```

6567 \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.

```

6568 \cs_set:Npn \@@_Hline_i:n #1
6569 {
6570   \peek_remove_spaces:n
6571   {
6572     \peek_meaning:NTF \Hline
6573     { \@@_Hline_ii:nn { #1 + 1 } }
6574     { \@@_Hline_iii:n { #1 } }
6575   }
6576 }
6577 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6578 \cs_set:Npn \@@_Hline_iii:n #1
6579 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6580 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6581 {
6582   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6583   \skip_vertical:N \l_@@_rule_width_dim
6584   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6585   {
6586     \@@_hline:n
6587     {
6588       multiplicity = #1 ,
6589       position = \int_eval:n { \c@iRow + 1 } ,

```

```

6590         total-width = \dim_use:N \l_@@_rule_width_dim ,
6591         #2
6592     }
6593 }
6594 \egroup
6595 }

```

### 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`).

```

6596 \cs_new_protected:Npn \@@_custom_line:n #1
6597 {
6598     \str_clear_new:N \l_@@_command_str
6599     \str_clear_new:N \l_@@_ccommand_str
6600     \str_clear_new:N \l_@@_letter_str
6601     \tl_clear_new:N \l_@@_other_keys_tl
6602     \keys_set_known:nn { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6603     \bool_lazy_all:nTF
6604     {
6605         { \str_if_empty_p:N \l_@@_letter_str }
6606         { \str_if_empty_p:N \l_@@_command_str }
6607         { \str_if_empty_p:N \l_@@_cccommand_str }
6608     }
6609     { \@@_error:n { No~letter~and~no~command } }
6610     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6611 }
6612 \keys_define:nn { nicematrix / custom-line }
6613 {
6614     letter .str_set:N = \l_@@_letter_str ,
6615     letter .value_required:n = true ,
6616     command .str_set:N = \l_@@_command_str ,
6617     command .value_required:n = true ,
6618     ccommand .str_set:N = \l_@@_cccommand_str ,
6619     ccommand .value_required:n = true ,
6620 }

```

```

6621 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6622 \cs_new_protected:Npn \@@_custom_line_i:n #1
6623 {

```

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

```

6624     \bool_set_false:N \l_@@_tikz_rule_bool
6625     \bool_set_false:N \l_@@_dotted_rule_bool
6626     \bool_set_false:N \l_@@_color_bool
6627     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6628     \bool_if:NT \l_@@_tikz_rule_bool
6629     {
6630         \IfPackageLoadedF { tikz }
6631         { \@@_error:n { tikz~in~custom-line~without~tikz } }
6632         \bool_if:NT \l_@@_color_bool
6633         { \@@_error:n { color~in~custom-line~with~tikz } }
6634     }

```

```

6635 \bool_if:NT \l_@@_dotted_rule_bool
6636 {
6637   \int_compare:nNt \l_@@_multiplicity_int > \c_one_int
6638   { \@@_error:n { key~multiplicity~with~dotted } }
6639 }
6640 \str_if_empty:NF \l_@@_letter_str
6641 {
6642   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6643   { \@@_error:n { Several~letters } }
6644   {
6645     \tl_if_in:NoTF
6646     \c_@@_forbidden_letters_str
6647     \l_@@_letter_str
6648     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6649   }

```

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.

```

6650   \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6651   { \@@_v_custom_line:n { #1 } }
6652 }
6653 }
6654 }
6655 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6656 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6657 }
6658 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6659 \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.

```

6660 \keys_define:nn { nicematrix / custom-line-bis }
6661 {
6662   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6663   multiplicity .initial:n = 1 ,
6664   multiplicity .value_required:n = true ,
6665   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6666   color .value_required:n = true ,
6667   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6668   tikz .value_required:n = true ,
6669   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6670   dotted .value_forbidden:n = true ,
6671   total-width .code:n = { } ,
6672   total-width .value_required:n = true ,
6673   width .code:n = { } ,
6674   width .value_required:n = true ,
6675   sep-color .code:n = { } ,
6676   sep-color .value_required:n = true ,
6677   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6678 }

```

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

```

6679 \bool_new:N \l_@@_dotted_rule_bool
6680 \bool_new:N \l_@@_tikz_rule_bool
6681 \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`.

```

6682 \keys_define:nn { nicematrix / custom-line-width }
6683 {
6684   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6685   multiplicity .initial:n = 1 ,
6686   multiplicity .value_required:n = true ,
6687   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6688   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6689   \bool_set_true:N \l_@@_total_width_bool ,
6690   total-width .value_required:n = true ,
6691   width .meta:n = { total-width = #1 } ,
6692   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6693 }

```

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).

```

6694 \cs_new_protected:Npn \@@_h_custom_line:n #1
6695 {

```

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).

```

6696   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6697   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6698 }

```

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).

```

6699 \cs_new_protected:Npn \@@_c_custom_line:n #1
6700 {

```

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

```

6701   \exp_args:Nc \NewExpandableDocumentCommand
6702   { nicematrix - \l_@@_ccommand_str }
6703   { 0 { } m }
6704   {
6705     \noalign
6706     {
6707       \@@_compute_rule_width:n { #1 , ##1 }
6708       \skip_vertical:n { \l_@@_rule_width_dim }
6709       \clist_map_inline:nn
6710       { ##2 }
6711       { \@@_c_custom_line_i:nn { #1 , ##1 } { ##### } }
6712     }
6713   }
6714   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6715 }

```

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*.

```

6716 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6717 {
6718   \tl_if_in:nnTF { #2 } { - }
6719   { \@@_cut_on_hyphen:w #2 \q_stop }
6720   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6721   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6722   {
6723     \@@_hline:n
6724     {
6725       #1 ,
6726       start = \l_tmpa_tl ,

```

```

6727         end = \l_tmpb_tl ,
6728         position = \int_eval:n { \c@iRow + 1 } ,
6729         total-width = \dim_use:N \l_@@_rule_width_dim
6730     }
6731 }
6732 }
6733 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6734 {
6735     \bool_set_false:N \l_@@_tikz_rule_bool
6736     \bool_set_false:N \l_@@_total_width_bool
6737     \bool_set_false:N \l_@@_dotted_rule_bool
6738     \keys_set_known:n { nicematrix / custom-line-width } { #1 }
6739     \bool_if:NF \l_@@_total_width_bool
6740     {
6741         \bool_if:NTF \l_@@_dotted_rule_bool
6742         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6743         {
6744             \bool_if:NF \l_@@_tikz_rule_bool
6745             {
6746                 \dim_set:Nn \l_@@_rule_width_dim
6747                 {
6748                     \arrayrulewidth * \l_@@_multiplicity_int
6749                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6750                 }
6751             }
6752         }
6753     }
6754 }
6755 \cs_new_protected:Npn \@@_v_custom_line:n #1
6756 {
6757     \@@_compute_rule_width:n { #1 }

```

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

```

6758     \tl_gput_right:Ne \g_@@_array_preamble_tl
6759     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6760     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6761     {
6762         \@@_vline:n
6763         {
6764             #1 ,
6765             position = \int_eval:n { \c@jCol + 1 } ,
6766             total-width = \dim_use:N \l_@@_rule_width_dim
6767         }
6768     }
6769     \@@_rec_preamble:n
6770 }
6771 \@@_custom_line:n
6772 { 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.

```

6773 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6774 {
6775     \int_compare:nNt \l_tmpa_tl > { #1 }
6776     {
6777         \int_compare:nNt \l_tmpa_tl < { #3 + 1 }
6778         {
6779             \int_compare:nNt \l_tmpb_tl > { #2 - 1 }

```

```

6780         {
6781             \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6782             { \bool_gset_false:N \g_tmpa_bool }
6783         }
6784     }
6785 }
6786 }

```

The same for vertical rules.

```

6787 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6788 {
6789     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6790     {
6791         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6792         {
6793             \int_compare:nNnT \l_tmpb_tl > { #2 }
6794             {
6795                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6796                 { \bool_gset_false:N \g_tmpa_bool }
6797             }
6798         }
6799     }
6800 }
6801 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6802 {
6803     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6804     {
6805         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6806         {
6807             \int_compare:nNnTF \l_tmpa_tl = { #1 }
6808             { \bool_gset_false:N \g_tmpa_bool }
6809             {
6810                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6811                 { \bool_gset_false:N \g_tmpa_bool }
6812             }
6813         }
6814     }
6815 }
6816 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6817 {
6818     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6819     {
6820         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6821         {
6822             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6823             { \bool_gset_false:N \g_tmpa_bool }
6824             {
6825                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6826                 { \bool_gset_false:N \g_tmpa_bool }
6827             }
6828         }
6829     }
6830 }

```

## 23 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.

```

6831 \cs_new_protected:Npn \@@_compute_corners:
6832 {
6833   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6834   { \@@_mark_cells_of_block:nnnnn ##1 }

```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

6835   \clist_clear:N \l_@@_corners_cells_clist
6836   \clist_map_inline:Nn \l_@@_corners_clist
6837   {
6838     \str_case:nnF { ##1 }
6839     {
6840       { NW }
6841       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6842       { NE }
6843       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6844       { SW }
6845       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6846       { SE }
6847       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6848     }
6849     { \@@_error:nn { bad~corner } { ##1 } }
6850   }

```

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

```

6851   \clist_if_empty:NF \l_@@_corners_cells_clist
6852   {

```

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 colors the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6853     \tl_gput_right:Ne \g_@@_aux_tl
6854     {
6855       \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6856       { \l_@@_corners_cells_clist }
6857     }
6858   }
6859 }

```

```

6860 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6861 {
6862   \int_step_inline:nnn { #1 } { #3 }
6863   {
6864     \int_step_inline:nnn { #2 } { #4 }
6865     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6866   }
6867 }

```

```

6868 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6869 {
6870   \cs_if_exist:cTF
6871   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6872   \prg_return_true:
6873   \prg_return_false:
6874 }

```

“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_clist`.

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.

```
6875 \cs_new_protected:Npn \@@_compute_a_corner:nnnnn #1 #2 #3 #4 #5 #6
6876 {
```

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.

```
6877 \bool_set_false:N \l_tmpa_bool
6878 \int_zero_new:N \l_@@_last_empty_row_int
6879 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6880 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6881 {
6882   \bool_lazy_or:nnTF
6883   {
6884     \cs_if_exist_p:c
6885     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6886   }
6887   { \@@_if_in_block_p:nn { ##1 } { #2 } }
6888   { \bool_set_true:N \l_tmpa_bool }
6889   {
6890     \bool_if:NF \l_tmpa_bool
6891     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6892   }
6893 }
```

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

```
6894 \bool_set_false:N \l_tmpa_bool
6895 \int_zero_new:N \l_@@_last_empty_column_int
6896 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6897 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6898 {
6899   \bool_lazy_or:nnTF
6900   {
6901     \cs_if_exist_p:c
6902     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6903   }
6904   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6905   { \bool_set_true:N \l_tmpa_bool }
6906   {
6907     \bool_if:NF \l_tmpa_bool
6908     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6909   }
6910 }
```

Now, we loop over the rows.

```
6911 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6912 {
```

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

```
6913 \bool_set_false:N \l_tmpa_bool
6914 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6915 {
6916   \bool_lazy_or:nnTF
6917   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6918   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6919   { \bool_set_true:N \l_tmpa_bool }
6920   {
6921     \bool_if:NF \l_tmpa_bool
```

```

6922         {
6923             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6924             \clist_put_right:Nn
6925                 \l_@@_corners_cells_clist
6926                 { ##1 - #####1 }
6927             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6928         }
6929     }
6930 }
6931 }
6932 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6933 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6934 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:  
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

## 24 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.

```

6935 \bool_new:N \l_@@_block_auto_columns_width_bool

```

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

```

6936 \keys_define:nn { nicematrix / NiceMatrixBlock }
6937 {
6938     auto-columns-width .code:n =
6939     {
6940         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6941         \dim_gzero_new:N \g_@@_max_cell_width_dim
6942         \bool_set_true:N \l_@@_auto_columns_width_bool
6943     }
6944 }

6945 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6946 {
6947     \int_gincr:N \g_@@_NiceMatrixBlock_int
6948     \dim_zero:N \l_@@_columns_width_dim
6949     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6950     \bool_if:NT \l_@@_block_auto_columns_width_bool
6951     {
6952         \cs_if_exist:cT
6953             { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6954         {
6955             \dim_set:Nn \l_@@_columns_width_dim
6956             {
6957                 \use:c
6958                     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6959             }
6960         }
6961     }
6962 }

```

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`).

```
6963 {
6964   \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.

```
6965   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6966   {
6967     \bool_if:NT \l_@@_block_auto_columns_width_bool
6968     {
6969       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6970       \iow_shipout:Ne \@mainaux
6971       {
6972         \cs_gset:cpn
6973         { @@ _ 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.

```
6974         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6975       }
6976       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6977     }
6978   }
6979   \ignorespacesafterend
6980 }
```

## 25 The extra nodes

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).

```
6981 \cs_new_protected:Npn \@@_create_extra_nodes:
6982 {
6983   \bool_if:nTF \l_@@_medium_nodes_bool
6984   {
6985     \bool_if:NTF \l_@@_no_cell_nodes_bool
6986     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6987     {
6988       \bool_if:NTF \l_@@_large_nodes_bool
6989       \@@_create_medium_and_large_nodes:
6990       \@@_create_medium_nodes:
6991     }
6992   }
6993   {
6994     \bool_if:NT \l_@@_large_nodes_bool
6995     {
6996       \bool_if:NTF \l_@@_no_cell_nodes_bool
6997       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6998       \@@_create_large_nodes:
6999     }
7000   }
7001 }
```

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`.

```

7002 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7003 {
7004   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7005   {
7006     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
7007     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
7008     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
7009     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
7010   }
7011   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7012   {
7013     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
7014     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
7015     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
7016     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
7017   }

```

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

```

7018   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7019   {
7020     \int_step_variable:nnNn
7021     \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.

```

7022     {
7023       \cs_if_exist:cT
7024       { 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`.

```

7025     {
7026       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7027       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
7028       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7029       \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7030       {
7031         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7032         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7033       }

```

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`.

```

7034       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7035       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7036       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7037       \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7038       {
7039         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

7040             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7041         }
7042     }
7043 }
7044 }

```

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

```

7045 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7046 {
7047     \dim_compare:nNnT
7048     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7049     {
7050         \@@_qpoint:n { row - \@@_i: - base }
7051         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7052         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7053     }
7054 }
7055 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7056 {
7057     \dim_compare:nNnT
7058     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7059     {
7060         \@@_qpoint:n { col - \@@_j: }
7061         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7062         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7063     }
7064 }
7065 }

```

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

```

7066 \cs_new_protected:Npn \@@_create_medium_nodes:
7067 {
7068     \pgfpicture
7069     \pgfrememberpicturepositiononpagetrue
7070     \pgf@relevantforpicturesizefalse
7071     \@@_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”.

```

7072     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7073     \@@_create_nodes:
7074     \endpgfpicture
7075 }

```

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:`.

```

7076 \cs_new_protected:Npn \@@_create_large_nodes:
7077 {
7078     \pgfpicture
7079     \pgfrememberpicturepositiononpagetrue
7080     \pgf@relevantforpicturesizefalse
7081     \@@_computations_for_medium_nodes:
7082     \@@_computations_for_large_nodes:
7083     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7084     \@@_create_nodes:

```

---

<sup>14</sup>If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7085 \endpgfpicture
7086 }
7087 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7088 {
7089 \pgfpicture
7090 \pgfrememberpicturepositiononpagetrue
7091 \pgf@relevantforpicturesizefalse
7092 \@@_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”.

```

7093 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
7094 \@@_create_nodes:
7095 \@@_computations_for_large_nodes:
7096 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7097 \@@_create_nodes:
7098 \endpgfpicture
7099 }

```

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.

```

7100 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7101 {
7102 \int_set_eq:NN \l_@@_first_row_int \c_one_int
7103 \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`.

```

7104 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7105 {
7106 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7107 {
7108 (
7109 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7110 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7111 )
7112 / 2
7113 }
7114 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7115 { l_@@_row _ \@@_i: _ min _ dim }
7116 }
7117 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7118 {
7119 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7120 {
7121 (
7122 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7123 \dim_use:c
7124 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7125 )
7126 / 2
7127 }
7128 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7129 { l_@@_column _ \@@_j: _ max _ dim }
7130 }

```

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

```

7131 \dim_sub:cn
7132 { l_@@_column _ 1 _ min _ dim }
7133 \l_@@_left_margin_dim
7134 \dim_add:cn
7135 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7136 \l_@@_right_margin_dim
7137 }

```

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`).

```

7138 \cs_new_protected:Npn \@@_create_nodes:
7139 {
7140   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7141   {
7142     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7143     {

```

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

```

7144       \@@_pgf_rect_node:nnnnn
7145       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7146       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7147       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7148       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7149       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7150       \str_if_empty:NF \l_@@_name_str
7151       {
7152         \pgfnodealias
7153         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7154         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7155       }
7156     }
7157   }

```

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  $n$ .

```

7158   \seq_map_pairwise_function:NNN
7159   \g_@@_multicolumn_cells_seq
7160   \g_@@_multicolumn_sizes_seq
7161   \@@_node_for_multicolumn:nn
7162 }

7163 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7164 {
7165   \cs_set_nopar:Npn \@@_i: { #1 }
7166   \cs_set_nopar:Npn \@@_j: { #2 }
7167 }

```

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”).

```

7168 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7169 {
7170   \@@_extract_coords_values: #1 \q_stop
7171   \@@_pgf_rect_node:nnnnn
7172   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7173   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7174   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7175   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: + #2 - 1 } _max_dim } }
7176   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7177   \str_if_empty:NF \l_@@_name_str
7178   {
7179     \pgfnodealias
7180     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7181     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7182   }
7183 }

```

## 26 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).

```

7184 \keys_define:nn { nicematrix / Block / FirstPass }
7185 {
7186   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7187           \bool_set_true:N \l_@@_p_block_bool ,
7188   j .value_forbidden:n = true ,
7189   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7190   l .value_forbidden:n = true ,
7191   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7192   r .value_forbidden:n = true ,
7193   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7194   c .value_forbidden:n = true ,
7195   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7196   L .value_forbidden:n = true ,
7197   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7198   R .value_forbidden:n = true ,
7199   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7200   C .value_forbidden:n = true ,
7201   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7202   t .value_forbidden:n = true ,
7203   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7204   T .value_forbidden:n = true ,
7205   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7206   b .value_forbidden:n = true ,
7207   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7208   B .value_forbidden:n = true ,
7209   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7210   m .value_forbidden:n = true ,
7211   v-center .meta:n = m ,
7212   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7213   p .value_forbidden:n = true ,
7214   color .code:n =
7215     \@@_color:n { #1 }
7216     \tl_set_rescan:Nnn
7217       \l_@@_draw_tl
7218       { \char_set_catcode_other:N ! }
7219       { #1 } ,
7220   color .value_required:n = true ,
7221   respect-arraystretch .code:n =
7222     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7223   respect-arraystretch .value_forbidden:n = true ,
7224 }

```

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.

```

7225 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7226 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7227 {

```

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).

```

7228   \peek_remove_spaces:n

```



```

7229 {
7230   \tl_if_blank:nTF { #2 }
7231   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7232   {
7233     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7234     \@@_Block_i_czech \@@_Block_i
7235     #2 \q_stop
7236   }
7237   { #1 } { #3 } { #4 }
7238 }
7239 }

```

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

```

7240 \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`.

```

7241 {
7242   \char_set_catcode_active:N -
7243   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7244 }

```

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.

```

7245 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7246 {

```

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).

```

7247   \bool_lazy_or:nnTF
7248   { \tl_if_blank_p:n { #1 } }
7249   { \str_if_eq_p:ee { * } { #1 } }
7250   { \int_set:Nn \l_tmpa_int { 100 } }
7251   { \int_set:Nn \l_tmpa_int { #1 } }
7252   \bool_lazy_or:nnTF
7253   { \tl_if_blank_p:n { #2 } }
7254   { \str_if_eq_p:ee { * } { #2 } }
7255   { \int_set:Nn \l_tmpb_int { 100 } }
7256   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7257   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7258   {
7259     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7260     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7261     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7262   }
7263   { \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.

```

7264   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }

```

```

7265 \tl_set:N \l_tmpa_tl
7266 {
7267   { \int_use:N \c@iRow }
7268   { \int_use:N \c@jCol }
7269   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7270   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7271 }

```

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}`.

We have different treatments when the key `p` is used and when the block is mono-column or mono-row, etc. That’s why we have several macros: `\@@_Block_iv:nnnnn`, `\@@_Block_v:nnnnn`, `\@@_Block_vi:nnnn`, etc. (the five arguments of those macros are provided by curryfication).

```

7272 \bool_set_false:N \l_tmpa_bool
7273 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7274 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7275 \bool_case:nF
7276 {
7277   \l_tmpa_bool { \@@_Block_vii:eennn }
7278   \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

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.

```

7279 \l_@@_X_bool { \@@_Block_v:eennn }
7280 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7281 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7282 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7283 }
7284 { \@@_Block_v:eennn }
7285 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7286 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. 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.

```

7287 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7288 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7289 {
7290   \int_gincr:N \g_@@_block_box_int
7291   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7292   {
7293     \tl_gput_right:N \g_@@_pre_code_after_tl
7294     {
7295       \@@_actually_diagbox:nnnnnn
7296       { \int_use:N \c@iRow }
7297       { \int_use:N \c@jCol }
7298       { \int_eval:n { \c@iRow + #1 - 1 } }
7299       { \int_eval:n { \c@jCol + #2 - 1 } }
7300       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7301       { \g_@@_row_style_tl \exp_not:n { ##2 } }

```

```

7302     }
7303   }
7304   \box_gclear_new:c
7305   { 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!

```

7306   \hbox_gset:cn
7307   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7308   {

```

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}`).

```

7309     \tl_if_empty:NTF \l_@@_color_tl
7310     { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7311     { \@@_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`.

```

7312     \int_compare:nNnT { #1 } = \c_one_int
7313     {
7314       \int_if_zero:nTF \c@iRow
7315       {

```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```

%\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7316     \cs_set_eq:NN \Block \@@_NullBlock:
7317     \l_@@_code_for_first_row_tl
7318   }
7319   {
7320     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7321     {
7322       \cs_set_eq:NN \Block \@@_NullBlock:
7323       \l_@@_code_for_last_row_tl
7324     }
7325   }
7326   \g_@@_row_style_tl
7327 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```
7328 \@@_reset_arraystretch:
7329 \dim_zero:N \extrarowheight
```

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

```
7330 #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.).

```
7331 \@@_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}`.

```
7332 \bool_if:NTF \l_@@_tabular_bool
7333 {
7334   \bool_lazy_all:nTF
7335   {
7336     { \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.

```
7337   { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7338   { ! \g_@@_rotate_bool }
7339 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7340 {
7341   \use:e
7342   {
```

The `\exp_not:N` is mandatory before `\begin`.

```
7343   \exp_not:N \begin { minipage }%
7344   [ \str_lowercase:o \l_@@_vpos_block_str ]
7345   { \l_@@_col_width_dim }
7346   \str_case:on \l_@@_hpos_block_str
7347   { c \centering r \raggedleft l \raggedright }
7348   }
7349   #5
7350   \end { minipage }
7351 }
```

In the other cases, we use a `{tabular}`.

```
7352 {
7353   \bool_if:NT \c_@@_testphase_table_bool
7354   { \tagpdfsetup { table / tagging = presentation } }
7355   \use:e
7356   {
7357     \exp_not:N \begin { tabular }%
7358     [ \str_lowercase:o \l_@@_vpos_block_str ]
7359     { @ { } \l_@@_hpos_block_str @ { } }
7360   }
7361   #5
7362   \end { tabular }
7363 }
7364 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7365 {
7366   \c_math_toggle_token
7367   \use:e
7368   {
```

```

7369         \exp_not:N \begin { array }%
7370         [ \str_lowercase:o \l_@@_vpos_block_str ]
7371         { @ { } \l_@@_hpos_block_str @ { } }
7372     }
7373     #5
7374     \end { array }
7375     \c_math_toggle_token
7376 }
7377 }

```

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 of the rotated box).

```

7378     \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.

```

7379     \int_compare:nNnT { #2 } = \c_one_int
7380     {
7381         \dim_gset:Nn \g_@@_blocks_wd_dim
7382         {
7383             \dim_max:nn
7384             \g_@@_blocks_wd_dim
7385             {
7386                 \box_wd:c
7387                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7388             }
7389         }
7390     }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position.

```

7391     \bool_lazy_and:nnT
7392     { \int_compare_p:nNn { #1 } = \c_one_int }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7393     { \str_if_empty_p:N \l_@@_vpos_block_str }
7394     {
7395         \dim_gset:Nn \g_@@_blocks_ht_dim
7396         {
7397             \dim_max:nn
7398             \g_@@_blocks_ht_dim
7399             {
7400                 \box_ht:c
7401                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7402             }
7403         }
7404         \dim_gset:Nn \g_@@_blocks_dp_dim
7405         {
7406             \dim_max:nn
7407             \g_@@_blocks_dp_dim
7408             {
7409                 \box_dp:c
7410                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7411             }
7412         }
7413     }
7414     \seq_gput_right:Ne \g_@@_blocks_seq
7415     {
7416         \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.

```

7417     {
7418         \exp_not:n { #3 } ,
7419         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7420         \bool_if:NT \g_@@_rotate_bool
7421         {
7422             \bool_if:NTF \g_@@_rotate_c_bool
7423             { m }
7424             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7425         }
7426     }
7427     {
7428         \box_use_drop:c
7429         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7430     }
7431 }
7432 \bool_set_false:N \g_@@_rotate_c_bool
7433 }

```

```

7434 \cs_new:Npn \@@_adjust_hpos_rotate:
7435 {
7436     \bool_if:NT \g_@@_rotate_bool
7437     {
7438         \str_set:Ne \l_@@_hpos_block_str
7439         {
7440             \bool_if:NTF \g_@@_rotate_c_bool
7441             { c }
7442             {
7443                 \str_case:onF \l_@@_vpos_block_str
7444                 { b l B l t r T r }
7445                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7446             }
7447         }
7448     }
7449 }

```

Despite its name the following command rotates the box of the block *but also does vertical ajustement of the baseline of the block*.

```

7450 \cs_new_protected:Npn \@@_rotate_box_of_block:
7451 {
7452     \box_grotate:cn
7453     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7454     { 90 }
7455     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7456     {
7457         \vbox_gset_top:cn
7458         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7459         {
7460             \skip_vertical:n { 0.8 ex }
7461             \box_use:c
7462             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7463         }
7464     }
7465     \bool_if:NT \g_@@_rotate_c_bool
7466     {
7467         \hbox_gset:cn
7468         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7469     {
7470         \c_math_toggle_token
7471         \vcenter
7472         {
7473             \box_use:c
7474             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7475         }
7476         \c_math_toggle_token
7477     }
7478 }
7479 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). 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:nnnnn`).

#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.

```

7480 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7481 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7482 {
7483     \seq_gput_right:Ne \g_@@_blocks_seq
7484     {
7485         \l_tmpa_tl
7486         { \exp_not:n { #3 } }
7487         {
7488             \bool_if:NTF \l_@@_tabular_bool
7489             {
7490                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7491     \@@_reset_arraystretch:
7492     \exp_not:n
7493     {
7494         \dim_zero:N \extrarowheight
7495         #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).

```

7496         \bool_if:NT \c_@@_testphase_table_bool
7497         { \tag_stop:n { table } }
7498         \use:e
7499         {
7500             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7501             { @ { } \l_@@_hpos_block_str @ { } }
7502         }
7503         #5
7504         \end { tabular }
7505     }
7506     \group_end:
7507 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7508     {
7509         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7510     \@@_reset_arraystretch:
7511     \exp_not:n
7512     {

```

```

7513         \dim_zero:N \extrarowheight
7514         #4
7515         \c_math_toggle_token
7516         \use:e
7517         {
7518             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7519             { @ { } \l_@@_hpos_block_str @ { } }
7520         }
7521         #5
7522         \end { array }
7523         \c_math_toggle_token
7524     }
7525     \group_end:
7526 }
7527 }
7528 }
7529 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7530 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7531 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7532 {
7533     \seq_gput_right:Ne \g_@@_blocks_seq
7534     {
7535         \l_tmpa_tl
7536         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7537         { { \exp_not:n { #4 #5 } } }
7538     }
7539 }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7540 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7541 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7542 {
7543     \seq_gput_right:Ne \g_@@_blocks_seq
7544     {
7545         \l_tmpa_tl
7546         { \exp_not:n { #3 } }
7547         { \exp_not:n { #4 #5 } }
7548     }
7549 }

```

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).

```

7550 \keys_define:nn { nicematrix / Block / SecondPass }
7551 {
7552     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7553     ampersand-in-blocks .default:n = true ,
7554     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7555     tikz .code:n =
7556         \IfPackageLoadedTF { tikz }
7557         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7558         { \@@_error:n { tikz~key~without~tikz } } ,
7559     tikz .value_required:n = true ,
7560     fill .code:n =
7561         \tl_set_rescan:Nnn

```



```

7562     \l_@@_fill_tl
7563     { \char_set_catcode_other:N ! }
7564     { #1 } ,
7565 fill .value_required:n = true ,
7566 opacity .tl_set:N = \l_@@_opacity_tl ,
7567 opacity .value_required:n = true ,
7568 draw .code:n =
7569     \tl_set_rescan:Nnn
7570     \l_@@_draw_tl
7571     { \char_set_catcode_other:N ! }
7572     { #1 } ,
7573 draw .default:n = default ,
7574 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7575 rounded-corners .default:n = 4 pt ,
7576 color .code:n =
7577     \@@_color:n { #1 }
7578     \tl_set_rescan:Nnn
7579     \l_@@_draw_tl
7580     { \char_set_catcode_other:N ! }
7581     { #1 } ,
7582 borders .clist_set:N = \l_@@_borders_clist ,
7583 borders .value_required:n = true ,
7584 hvlines .meta:n = { vl_lines , hl_lines } ,
7585 vl_lines .bool_set:N = \l_@@_vl_lines_block_bool ,
7586 vl_lines .default:n = true ,
7587 hl_lines .bool_set:N = \l_@@_hl_lines_block_bool ,
7588 hl_lines .default:n = true ,
7589 line-width .dim_set:N = \l_@@_line_width_dim ,
7590 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7591 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7592     \bool_set_true:N \l_@@_p_block_bool ,
7593 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7594 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7595 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7596 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7597     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7598 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7599     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7600 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7601     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7602 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7603 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7604 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7605 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7606 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7607 m .value_forbidden:n = true ,
7608 v-center .meta:n = m ,
7609 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7610 p .value_forbidden:n = true ,
7611 name .tl_set:N = \l_@@_block_name_str ,
7612 name .value_required:n = true ,
7613 name .initial:n = ,
7614 respect-arraystretch .code:n =
7615     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7616 respect-arraystretch .value_forbidden:n = true ,
7617 transparent .bool_set:N = \l_@@_transparent_bool ,
7618 transparent .default:n = true ,
7619 transparent .initial:n = false ,
7620 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7621 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construc-

tion 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.

```

7622 \cs_new_protected:Npn \@@_draw_blocks:
7623 {
7624   \bool_if:NTF \c_@@_recent_array_bool
7625     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7626     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7627   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7628 }
7629 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7630 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7631   \int_zero_new:N \l_@@_last_row_int
7632   \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 follows: 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 (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7633   \int_compare:nNnTF { #3 } > { 98 }
7634     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7635     { \int_set:Nn \l_@@_last_row_int { #3 } }
7636   \int_compare:nNnTF { #4 } > { 98 }
7637     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7638     { \int_set:Nn \l_@@_last_col_int { #4 } }
7639   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7640     {
7641       \bool_lazy_and:nnTF
7642         \l_@@_preamble_bool
7643         {
7644           \int_compare_p:n
7645             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7646         }
7647         {
7648           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7649           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7650           \@@_msg_redirect_name:nn { columns-not-used } { none }
7651         }
7652       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7653     }
7654   {
7655     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7656       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7657       {
7658         \@@_Block_v:nneenn
7659         { #1 }
7660         { #2 }
7661         { \int_use:N \l_@@_last_row_int }
7662         { \int_use:N \l_@@_last_col_int }
7663         { #5 }
7664         { #6 }
7665       }
7666     }
7667   }

```

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

```

7668 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7669 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7670 {

```

The group is for the keys.

```

7671 \group_begin:
7672 \int_compare:nNt { #1 } = { #3 }
7673 { \str_set:Nn \l_@@_vpos_block_str { t } }
7674 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7675 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7676 \bool_lazy_and:nnT
7677 \l_@@_vlines_block_bool
7678 { ! \l_@@_ampersand_bool }
7679 {
7680 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7681 {
7682 \@@_vlines_block:nnn
7683 { \exp_not:n { #5 } }
7684 { #1 - #2 }
7685 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7686 }
7687 }
7688 \bool_if:NT \l_@@_hlines_block_bool
7689 {
7690 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7691 {
7692 \@@_hlines_block:nnn
7693 { \exp_not:n { #5 } }
7694 { #1 - #2 }
7695 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7696 }
7697 }
7698 \bool_if:NF \l_@@_transparent_bool
7699 {
7700 \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7701 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7702 \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7703 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7704 }
7705 }

```

```

7706 \tl_if_empty:NF \l_@@_draw_tl
7707 {
7708 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7709 { \@@_error:n { hlines~with~color } }
7710 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7711 {
7712 \@@_stroke_block:nnn

```

**#5** are the options

```

7713 { \exp_not:n { #5 } }
7714 { #1 - #2 }
7715 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7716 }
7717 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7718 { { #1 } { #2 } { #3 } { #4 } }
7719 }

```

```

7720 \clist_if_empty:NF \l_@@_borders_clist
7721 {
7722   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7723   {
7724     \@@_stroke_borders_block:nnn
7725     { \exp_not:n { #5 } }
7726     { #1 - #2 }
7727     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7728   }
7729 }
7730 \tl_if_empty:NF \l_@@_fill_tl
7731 {
7732   \@@_add_opacity_to_fill:
7733   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7734   {
7735     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7736     { #1 - #2 }
7737     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7738     { \dim_use:N \l_@@_rounded_corners_dim }
7739   }
7740 }
7741 \seq_if_empty:NF \l_@@_tikz_seq
7742 {
7743   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7744   {
7745     \@@_block_tikz:nnnnn
7746     { \seq_use:Nn \l_@@_tikz_seq { , } }
7747     { #1 }
7748     { #2 }
7749     { \int_use:N \l_@@_last_row_int }
7750     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7751   }
7752 }
7753 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7754 {
7755   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7756   {
7757     \@@_actually_diagbox:nnnnnn
7758     { #1 }
7759     { #2 }
7760     { \int_use:N \l_@@_last_row_int }
7761     { \int_use:N \l_@@_last_col_int }
7762     { \exp_not:n { ##1 } }
7763     { \exp_not:n { ##2 } }
7764   }
7765 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & four & five & \\\
six & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7766 \pgfpicture
7767 \pgfrememberpicturepositiononpagetrue
7768 \pgf@relevantforpicturesizefalse
7769 \@@_qpoint:n { row - #1 }
7770 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7771 \@@_qpoint:n { col - #2 }
7772 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7773 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7774 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7775 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7776 \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.

```

7777 \@@_pgf_rect_node:nnnnn
7778 { \@@_env: - #1 - #2 - block }
7779 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7780 \str_if_empty:NF \l_@@_block_name_str
7781 {
7782   \pgfnodealias
7783   { \@@_env: - \l_@@_block_name_str }
7784   { \@@_env: - #1 - #2 - block }
7785   \str_if_empty:NF \l_@@_name_str
7786   {
7787     \pgfnodealias
7788     { \l_@@_name_str - \l_@@_block_name_str }
7789     { \@@_env: - #1 - #2 - block }
7790   }
7791 }

```

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.

```

7792 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7793 {
7794   \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.

```

7795 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7796 {

```

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.

```

7797 \cs_if_exist:cT
7798 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7799 {
7800   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7801   {
7802     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7803     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7804   }
7805 }
7806 }

```

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.

```

7807     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7808     {
7809         \@@_qpoint:n { col - #2 }
7810         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7811     }
7812     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7813     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7814     {
7815         \cs_if_exist:cT
7816         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7817         {
7818             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7819             {
7820                 \pgfpointanchor
7821                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7822                 { east }
7823                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7824             }
7825         }
7826     }
7827     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7828     {
7829         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7830         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7831     }
7832     \@@_pgf_rect_node:nnnnn
7833     { \@@_env: - #1 - #2 - block - short }
7834     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7835 }

```

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.

```

7836     \bool_if:NT \l_@@_medium_nodes_bool
7837     {
7838         \@@_pgf_rect_node:nnn
7839         { \@@_env: - #1 - #2 - block - medium }
7840         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7841         {
7842             \pgfpointanchor
7843             { \@@_env:
7844                 - \int_use:N \l_@@_last_row_int
7845                 - \int_use:N \l_@@_last_col_int - medium
7846             }
7847             { south-east }
7848         }
7849     }
7850     \endpgfpicture

7851     \bool_if:NTF \l_@@_ampersand_bool
7852     {
7853         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7854         \int_zero_new:N \l_@@_split_int
7855         \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7856         \pgfpicture
7857         \pgfrememberpicturepositiononpagetrue
7858         \pgf@relevantforpicturesizefalse
7859
7860         \@@_qpoint:n { row - #1 }
7861         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7862         \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }

```

```

7863 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7864 \@@_qpoint:n { col - #2 }
7865 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7866 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7867 \dim_set:Nn \l_tmpb_dim
7868 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7869 \bool_lazy_or:nnT
7870 \l_@@_vlines_block_bool
7871 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7872 {
7873   \int_step_inline:nn { \l_@@_split_int - 1 }
7874   {
7875     \pgfpathmoveto
7876     {
7877       \pgfpoint
7878       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7879       \l_@@_tmpc_dim
7880     }
7881     \pgfpathlineto
7882     {
7883       \pgfpoint
7884       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7885       \l_@@_tmpd_dim
7886     }
7887     \CT@arc@
7888     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7889     \pgfsetrectcap
7890     \pgfusepathqstroke
7891   }
7892 }
7893 \@@_qpoint:n { row - #1 - base }
7894 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7895 \int_step_inline:nn \l_@@_split_int
7896 {
7897   \group_begin:
7898   \dim_set:Nn \col@sep
7899   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7900   \pgftransformshift
7901   {
7902     \pgfpoint
7903     {
7904       \l_tmpa_dim + ##1 \l_tmpb_dim -
7905       \str_case:on \l_@@_hpos_block_str
7906       {
7907         l { \l_tmpb_dim + \col@sep }
7908         c { 0.5 \l_tmpb_dim }
7909         r { \col@sep }
7910       }
7911     }
7912     { \l_@@_tmpc_dim }
7913   }
7914   \pgfset { inner~sep = \c_zero_dim }
7915   \pgfnode
7916   { rectangle }
7917   {
7918     \str_case:on \l_@@_hpos_block_str
7919     {
7920       c { base }
7921       l { base~west }
7922       r { base~east }
7923     }
7924   }
7925   { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }

```

```

7926         \group_end:
7927     }
7928     \endpgfpicture
7929 }

```

Now the case where there is no ampersand & in the content of the block.

```

7930 {
7931     \bool_if:NTF \l_@@_p_block_bool
7932     {

```

When the final user has used the key p, we have to compute the width.

```

7933         \pgfpicture
7934         \pgfrememberpicturepositiononpagetrue
7935         \pgf@relevantforpicturesizefalse
7936         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7937         {
7938             \@@_qpoint:n { col - #2 }
7939             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7940             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7941         }
7942         {
7943             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7944             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7945             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7946         }
7947         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7948     \endpgfpicture
7949     \hbox_set:Nn \l_@@_cell_box
7950     {
7951         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7952         { \g_tmpb_dim }
7953         \str_case:on \l_@@_hpos_block_str
7954         { c \centering r \raggedleft l \raggedright j { } }
7955         #6
7956         \end { minipage }
7957     }
7958 }
7959 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7960 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Now, we will put the label of the block. We recall that \l\_@@\_vpos\_block\_str is empty when the user has not used a key for the vertical position of the block.

```

7961     \pgfpicture
7962     \pgfrememberpicturepositiononpagetrue
7963     \pgf@relevantforpicturesizefalse
7964     \bool_lazy_any:nTF
7965     {
7966         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7967         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7968         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7969         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7970     }
7971     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7972         \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.

```

7973         \bool_if:nT \g_@@_last_col_found_bool
7974         {
7975             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7976             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7977         }

```



`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7978      \tl_set:Ne \l_tmpa_tl
7979      {
7980          \str_case:on \l_@@_vpos_block_str
7981          {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7982      { } { % added 2024-06-29
7983          \str_case:on \l_@@_hpos_block_str
7984          {
7985              c { center }
7986              l { west }
7987              r { east }
7988              j { center }
7989          }
7990      }
7991      c {
7992          \str_case:on \l_@@_hpos_block_str
7993          {
7994              c { center }
7995              l { west }
7996              r { east }
7997              j { center }
7998          }
7999      }
8000  }
8001  T {
8002      \str_case:on \l_@@_hpos_block_str
8003      {
8004          c { north }
8005          l { north-west }
8006          r { north-east }
8007          j { north }
8008      }
8009  }
8010  }
8011  B {
8012      \str_case:on \l_@@_hpos_block_str
8013      {
8014          c { south }
8015          l { south-west }
8016          r { south-east }
8017          j { south }
8018      }
8019  }
8020  }
8021  }
8022  }
8023  \pgftransformshift
8024  {
8025      \pgfpointanchor
8026      {
8027          \@@_env: - #1 - #2 - block
8028          \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8029      }
8030      { \l_tmpa_tl }
8031  }
8032  \pgfset { inner~sep = \c_zero_dim }
8033  \pgfnode
8034  { rectangle }
8035  { \l_tmpa_tl }
8036  { \box_use_drop:N \l_@@_cell_box } { } { }

```

```
8037     }
```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```
8038     {
8039         \pgfextracty \l_tmpa_dim
8040         {
8041             \l_@@_qpoint:n
8042             {
8043                 row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8044                 - base
8045             }
8046         }
8047         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the  $x$ -value of the center of the block.

```
8048         \pgfpointanchor
8049         {
8050             \l_@@_env: - #1 - #2 - block
8051             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8052         }
8053         {
8054             \str_case:on \l_@@_hpos_block_str
8055             {
8056                 c { center }
8057                 l { west }
8058                 r { east }
8059                 j { center }
8060             }
8061         }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
8062         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8063         \pgfset { inner~sep = \c_zero_dim }
8064         \pgfnode
8065         { rectangle }
8066         {
8067             \str_case:on \l_@@_hpos_block_str
8068             {
8069                 c { base }
8070                 l { base~west }
8071                 r { base~east }
8072                 j { base }
8073             }
8074         }
8075         { \box_use_drop:N \l_@@_cell_box } { } { }
8076     }
8077     \endpgfpicture
8078 }
8079 \group_end:
8080 }
```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```
8081 \cs_set_protected:Npn \l_@@_fill:nnnnn #1 #2 #3 #4 #5
8082 {
8083     \pgfpicture
8084     \pgfrememberpicturepositiononpagetrue
8085     \pgf@relevantforpicturesizefalse
8086     \pgfpathrectanglecorners
8087     { \pgfpoint { #2 } { #3 } }
8088     { \pgfpoint { #4 } { #5 } }
8089     \pgfsetfillcolor { #1 }
8090     \pgfusepath { fill }
```

```

8091 \endpgfpicture
8092 }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8093 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8094 {
8095   \tl_if_empty:NF \l_@@_opacity_tl
8096   {
8097     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8098       {
8099         \tl_set:Nc \l_@@_fill_tl
8100         {
8101           [ opacity = \l_@@_opacity_tl ,
8102             \tl_tail:o \l_@@_fill_tl
8103         }
8104       }
8105       {
8106         \tl_set:Nc \l_@@_fill_tl
8107         { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8108       }
8109     }
8110   }

```

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).

```

8111 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8112 {
8113   \group_begin:
8114   \tl_clear:N \l_@@_draw_tl
8115   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8116   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8117   \pgfpicture
8118   \pgfrememberpicturepositiononpagetrue
8119   \pgf@relevantforpicturesizefalse
8120   \tl_if_empty:NF \l_@@_draw_tl
8121   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8122     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8123     { \CT@arc@ }
8124     { \@@_color:o \l_@@_draw_tl }
8125   }
8126   \pgfsetcornersarced
8127   {
8128     \pgfpoint
8129     { \l_@@_rounded_corners_dim }
8130     { \l_@@_rounded_corners_dim }
8131   }
8132   \@@_cut_on_hyphen:w #2 \q_stop
8133   \int_compare:nNf \l_tmpa_tl > \c@iRow
8134   {
8135     \int_compare:nNf \l_tmpb_tl > \c@jCol
8136     {
8137       \@@_qpoint:n { row - \l_tmpa_tl }
8138       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8139       \@@_qpoint:n { col - \l_tmpb_tl }
8140       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8141       \@@_cut_on_hyphen:w #3 \q_stop

```

```

8142 \int_compare:nNnT \l_tmpa_tl > \c@iRow
8143 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8144 \int_compare:nNnT \l_tmpb_tl > \c@jCol
8145 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8146 @@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8147 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8148 @@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8149 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8150 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8151 \pgfpathrectanglecorners
8152 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8153 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8154 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8155 { \pgfusepathqstroke }
8156 { \pgfusepath { stroke } }
8157 }
8158 }
8159 \endpgfpicture
8160 \group_end:
8161 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8162 \keys_define:nn { nicematrix / BlockStroke }
8163 {
8164   color .tl_set:N = \l_@@_draw_tl ,
8165   draw .code:n =
8166     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8167   draw .default:n = default ,
8168   line-width .dim_set:N = \l_@@_line_width_dim ,
8169   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8170   rounded-corners .default:n = 4 pt
8171 }

```

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).

```

8172 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8173 {
8174   \group_begin:
8175   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8176   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8177   @@_cut_on_hyphen:w #2 \q_stop
8178   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8179   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8180   @@_cut_on_hyphen:w #3 \q_stop
8181   \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8182   \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8183   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8184   {
8185     \use:e
8186     {
8187       @@_vline:n
8188       {
8189         position = ##1 ,
8190         start = \l_@@_tmpc_tl ,
8191         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8192         total-width = \dim_use:N \l_@@_line_width_dim
8193       }
8194     }
8195   }
8196   \group_end:
8197 }

```

```

8198 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8199 {
8200   \group_begin:
8201   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8202   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8203   \@@_cut_on_hyphen:w #2 \q_stop
8204   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8205   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8206   \@@_cut_on_hyphen:w #3 \q_stop
8207   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8208   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8209   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8210   {
8211     \use:e
8212     {
8213       \@@_hline:n
8214       {
8215         position = ##1 ,
8216         start = \l_@@_tmpd_tl ,
8217         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8218         total-width = \dim_use:N \l_@@_line_width_dim
8219       }
8220     }
8221   }
8222   \group_end:
8223 }

```

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).

```

8224 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8225 {
8226   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8227   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8228   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8229   { \@@_error:n { borders~forbidden } }
8230   {
8231     \tl_clear_new:N \l_@@_borders_tikz_tl
8232     \keys_set:no
8233     { nicematrix / OnlyForTikzInBorders }
8234     \l_@@_borders_clist
8235     \@@_cut_on_hyphen:w #2 \q_stop
8236     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8237     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8238     \@@_cut_on_hyphen:w #3 \q_stop
8239     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8240     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8241     \@@_stroke_borders_block_i:
8242   }
8243 }
8244 \hook_gput_code:nnn { begindocument } { . }
8245 {
8246   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8247   {
8248     \c_@@_pgfortikzpicture_tl
8249     \@@_stroke_borders_block_ii:
8250     \c_@@_endpgfortikzpicture_tl
8251   }
8252 }
8253 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8254 {
8255   \pgfrememberpicturepositiononpagetrue

```

```

8256 \pgf@relevantforpicturesizefalse
8257 \CT@arc@
8258 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8259 \clist_if_in:NnT \l_@@_borders_clist { right }
8260 { \@@_stroke_vertical:n \l_tmpb_tl }
8261 \clist_if_in:NnT \l_@@_borders_clist { left }
8262 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8263 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8264 { \@@_stroke_horizontal:n \l_tmpa_tl }
8265 \clist_if_in:NnT \l_@@_borders_clist { top }
8266 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8267 }
8268 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8269 {
8270   tikz .code:n =
8271     \cs_if_exist:NTF \tikzpicture
8272     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8273     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8274   tikz .value_required:n = true ,
8275   top .code:n = ,
8276   bottom .code:n = ,
8277   left .code:n = ,
8278   right .code:n = ,
8279   unknown .code:n = \@@_error:n { bad~border }
8280 }

```

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).

```

8281 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8282 {
8283   \@@_qpoint:n \l_@@_tmpc_tl
8284   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8285   \@@_qpoint:n \l_tmpa_tl
8286   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8287   \@@_qpoint:n { #1 }
8288   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8289   {
8290     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8291     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8292     \pgfusepath{stroke}
8293   }
8294   {
8295     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8296     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8297   }
8298 }

```

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).

```

8299 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8300 {
8301   \@@_qpoint:n \l_@@_tmpd_tl
8302   \clist_if_in:NnTF \l_@@_borders_clist { left }
8303   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8304   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8305   \@@_qpoint:n \l_tmpb_tl
8306   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8307   \@@_qpoint:n { #1 }
8308   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8309   {
8310     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8311     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8312     \pgfusepath{stroke}

```

```

8313     }
8314     {
8315         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8316         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8317     }
8318 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8319 \keys_define:nn { nicematrix / BlockBorders }
8320 {
8321     borders .clist_set:N = \l_@@_borders_clist ,
8322     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8323     rounded-corners .default:n = 4 pt ,
8324     line-width .dim_set:N = \l_@@_line_width_dim
8325 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`.

`#1` is a *list of lists* of Tikz keys used with the path.

*Example:* `{{offset=1pt,draw,red},{offset=2pt,draw,blue}}`

which arises from a command such as :

`\Block[tikz={offset=1pt,draw,red},tikz={offset=2pt,draw,blue}]{2-2}{}`

The arguments `#2` and `#3` are the coordinates of the first cell and `#4` and `#5` the coordinates of the last cell of the block.

```

8326 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8327 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8328 {
8329     \begin { tikzpicture }
8330     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8331     \clist_map_inline:nn { #1 }
8332     {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8333         \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8334         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8335         (
8336             [
8337                 xshift = \dim_use:N \l_@@_offset_dim ,
8338                 yshift = - \dim_use:N \l_@@_offset_dim
8339             ]
8340             #2 -| #3
8341         )
8342         rectangle
8343         (
8344             [
8345                 xshift = - \dim_use:N \l_@@_offset_dim ,
8346                 yshift = \dim_use:N \l_@@_offset_dim
8347             ]
8348             \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8349         ) ;
8350     }
8351 \end { tikzpicture }
8352 }

```

```

8353 \keys_define:nn { nicematrix / SpecialOffset }
8354 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8355 \cs_new_protected:Npn \@@_NullBlock:
8356 { \@@_collect_options:n { \@@_NullBlock_i: } }
8357 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8358 { }

```

## 27 How to draw the dotted lines transparently

```

8359 \cs_set_protected:Npn \@@_renew_matrix:
8360 {
8361   \RenewDocumentEnvironment { pmatrix } { } {
8362     { \pNiceMatrix }
8363     { \endpNiceMatrix }
8364   \RenewDocumentEnvironment { vmatrix } { } {
8365     { \vNiceMatrix }
8366     { \endvNiceMatrix }
8367   \RenewDocumentEnvironment { Vmatrix } { } {
8368     { \VNiceMatrix }
8369     { \endVNiceMatrix }
8370   \RenewDocumentEnvironment { bmatrix } { } {
8371     { \bNiceMatrix }
8372     { \endbNiceMatrix }
8373   \RenewDocumentEnvironment { Bmatrix } { } {
8374     { \BNiceMatrix }
8375     { \endBNiceMatrix }
8376 }

```

## 28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8377 \keys_define:nn { nicematrix / Auto }
8378 {
8379   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8380   columns-type .value_required:n = true ,
8381   l .meta:n = { columns-type = l } ,
8382   r .meta:n = { columns-type = r } ,
8383   c .meta:n = { columns-type = c } ,
8384   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8385   delimiters / color .value_required:n = true ,
8386   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8387   delimiters / max-width .default:n = true ,
8388   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8389   delimiters .value_required:n = true ,
8390   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8391   rounded-corners .default:n = 4 pt
8392 }
8393 \NewDocumentCommand \AutoNiceMatrixWithDelims
8394 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8395 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8396 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8397 {

```

The group is for the protection of the keys.

```

8398 \group_begin:
8399 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8400 \use:e
8401 {
8402   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8403     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }

```



```

8404     [ \exp_not:o \l_tmpa_tl ]
8405   }
8406   \int_if_zero:nT \l_@@_first_row_int
8407   {
8408     \int_if_zero:nT \l_@@_first_col_int { & }
8409     \prg_replicate:nn { #4 - 1 } { & }
8410     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8411   }
8412   \prg_replicate:nn { #3 }
8413   {
8414     \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).

```

8415     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8416     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8417   }
8418   \int_compare:nNnT \l_@@_last_row_int > { -2 }
8419   {
8420     \int_if_zero:nT \l_@@_first_col_int { & }
8421     \prg_replicate:nn { #4 - 1 } { & }
8422     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8423   }
8424   \end { NiceArrayWithDelims }
8425   \group_end:
8426 }
8427 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8428 {
8429   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8430   {
8431     \bool_gset_true:N \g_@@_delims_bool
8432     \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8433     \AutoNiceMatrixWithDelims { #2 } { #3 }
8434   }
8435 }
8436 \@@_define_com:nnn p ( )
8437 \@@_define_com:nnn b [ ]
8438 \@@_define_com:nnn v | |
8439 \@@_define_com:nnn V \ | \ |
8440 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8441 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8442 {
8443   \group_begin:
8444   \bool_gset_false:N \g_@@_delims_bool
8445   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8446   \group_end:
8447 }

```

## 29 The redefinition of the command \dotfill

```

8448 \cs_set_eq:NN \@@_old_dotfill \dotfill
8449 \cs_new_protected:Npn \@@_dotfill:
8450 {

```

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}).

```

8451 \@@_old_dotfill
8452 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8453 }

```

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`.

```

8454 \cs_new_protected:Npn \@@_dotfill_i:
8455 { \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

## 30 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.

```

8456 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8457 {
8458   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8459   {
8460     \@@_actually_diagbox:nnnnnn
8461     { \int_use:N \c@iRow }
8462     { \int_use:N \c@jCol }
8463     { \int_use:N \c@iRow }
8464     { \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.

```

8465   { \g_@@_row_style_tl \exp_not:n { #1 } }
8466   { \g_@@_row_style_tl \exp_not:n { #2 } }
8467 }

```

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`.

```

8468   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8469   {
8470     { \int_use:N \c@iRow }
8471     { \int_use:N \c@jCol }
8472     { \int_use:N \c@iRow }
8473     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8474     { }
8475   }
8476 }

```

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.

```

8477 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8478 {
8479   \pgfpicture
8480   \pgf@relevantforpicturesizefalse
8481   \pgfrememberpicturepositiononpagetrue
8482   \@@_qpoint:n { row - #1 }
8483   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8484   \@@_qpoint:n { col - #2 }
8485   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8486   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }

```

```

8487 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8488 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8489 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8490 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8491 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8492 {

```

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.

```

8493 \CT@arc@
8494 \pgfsetroundcap
8495 \pgfusepathqstroke
8496 }
8497 \pgfset { inner~sep = 1 pt }
8498 \pgfscope
8499 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8500 \pgfnode { rectangle } { south~west }
8501 {
8502 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```

8503 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8504 \end { minipage }
8505 }
8506 { }
8507 { }
8508 \endpgfscope
8509 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8510 \pgfnode { rectangle } { north~east }
8511 {
8512 \begin { minipage } { 20 cm }
8513 \raggedleft
8514 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8515 \end { minipage }
8516 }
8517 { }
8518 { }
8519 \endpgfpicture
8520 }

```

## 31 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. 83.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8521 \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 `\`.

```

8522 \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`.

```

8523 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8524 {
8525 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }

```

```

8526 \@@_CodeAfter_iv:n
8527 }

```

We catch the argument of the command `\end` (in `#1`).

```

8528 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8529 {

```

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.

```

8530 \str_if_eq:eeTF \@@_currenvir { #1 }
8531 { \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`.

```

8532 {
8533 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8534 \@@_CodeAfter_ii:n
8535 }
8536 }

```

## 32 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.

```

8537 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8538 {
8539 \pgfpicture
8540 \pgfrememberpicturepositiononpagetrue
8541 \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.

```

8542 \@@_qpoint:n { row - 1 }
8543 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8544 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8545 \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).

```

8546 \bool_if:nTF { #3 }
8547 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8548 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8549 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8550 {
8551 \cs_if_exist:cT
8552 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8553 {
8554 \pgfpointanchor
8555 { \@@_env: - ##1 - #2 }
8556 { \bool_if:nTF { #3 } { west } { east } }
8557 \dim_set:Nn \l_tmpa_dim

```

```

8558         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf{x }
8559     }
8560 }

```

Now we can put the delimiter with a node of PGF.

```

8561 \pgfset { inner~sep = \c_zero_dim }
8562 \dim_zero:N \nulldelimiterspace
8563 \pgftransformshift
8564 {
8565     \pgfpoint
8566     { \l_tmpa_dim
8567       { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8568     }
8569 \pgfnode
8570 { rectangle }
8571 { \bool_if:nTF { #3 } { east } { west } }
8572 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8573     \nullfont
8574     \c_math_toggle_token
8575     \@@_color:o \l_@@_delimiters_color_tl
8576     \bool_if:nTF { #3 } { \left #1 } { \left . }
8577     \vcenter
8578     {
8579         \nullfont
8580         \hrule \@height
8581             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8582             \@depth \c_zero_dim
8583             \@width \c_zero_dim
8584     }
8585     \bool_if:nTF { #3 } { \right . } { \right #1 }
8586     \c_math_toggle_token
8587 }
8588 { }
8589 { }
8590 \endpgfpicture
8591 }

```

### 33 The command \SubMatrix

```

8592 \keys_define:nn { nicematrix / sub-matrix }
8593 {
8594     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8595     extra-height .value_required:n = true ,
8596     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8597     left-xshift .value_required:n = true ,
8598     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8599     right-xshift .value_required:n = true ,
8600     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8601     xshift .value_required:n = true ,
8602     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8603     delimiters / color .value_required:n = true ,
8604     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8605     slim .default:n = true ,
8606     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8607     hlines .default:n = all ,
8608     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8609     vlines .default:n = all ,
8610     hvlines .meta:n = { hlines, vlines } ,
8611     hvlines .value_forbidden:n = true

```

```

8612 }
8613 \keys_define:nn { nicematrix }
8614 {
8615   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8616   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8617   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8618   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8619 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8620 \keys_define:nn { nicematrix / SubMatrix }
8621 {
8622   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8623   delimiters / color .value_required:n = true ,
8624   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8625   hlines .default:n = all ,
8626   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8627   vlines .default:n = all ,
8628   hvlines .meta:n = { hlines, vlines } ,
8629   hvlines .value_forbidden:n = true ,
8630   name .code:n =
8631     \tl_if_empty:nTF { #1 }
8632     { \@@_error:n { Invalid-name } }
8633     {
8634       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8635       {
8636         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8637         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8638         {
8639           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8640           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8641         }
8642       }
8643       { \@@_error:n { Invalid-name } }
8644     } ,
8645   name .value_required:n = true ,
8646   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8647   rules .value_required:n = true ,
8648   code .tl_set:N = \l_@@_code_tl ,
8649   code .value_required:n = true ,
8650   unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8651 }

8652 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8653 {
8654   \peek_remove_spaces:n
8655   {
8656     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8657     {
8658       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8659       [
8660         delimiters / color = \l_@@_delimiters_color_tl ,
8661         hlines = \l_@@_submatrix_hlines_clist ,
8662         vlines = \l_@@_submatrix_vlines_clist ,
8663         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8664         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8665         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8666         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8667         #5
8668       ]
8669     }
8670     \@@_SubMatrix_in_code_before_i { #2 } { #3 }

```

```

8671     }
8672 }
8673 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8674 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8675 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8676 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8677 {
8678   \seq_gput_right:Ne \g_@@_submatrix_seq
8679   {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8680     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8681     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8682     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8683     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8684   }
8685 }

```

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.

```

8686 \hook_gput_code:nnn { begindocument } { . }
8687 {
8688   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8689   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8690   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8691   {
8692     \peek_remove_spaces:n
8693     {
8694       \@@_sub_matrix:nnnnnnn
8695       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8696     }
8697   }
8698 }

```

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).

```

8699 \NewDocumentCommand \@@_compute_i_j:nn
8700 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8701 { \@@_compute_i_j:nnnn #1 #2 }
8702 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8703 {
8704   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8705   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8706   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8707   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8708   \tl_if_eq:NnT \l_@@_first_i_tl { last }

```

```

8709     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8710 \tl_if_eq:NnT \l_@@_first_j_tl { last }
8711     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8712 \tl_if_eq:NnT \l_@@_last_i_tl { last }
8713     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8714 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8715     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8716 }
8717 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8718 {
8719     \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8720     \@@_compute_i_j:nn { #2 } { #3 }
8721 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8722     { \cs_set_nopar:Npn \arraystretch { 1 } }
8723 \bool_lazy_or:nnTF
8724     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8725     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8726     { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8727 {
8728     \str_clear_new:N \l_@@_submatrix_name_str
8729 \keys_set:nn { nicematrix / SubMatrix } { #5 }
8730 \pgfpicture
8731 \pgfrememberpicturepositiononpagetrue
8732 \pgf@relevantforpicturesizefalse
8733 \pgfset { inner~sep = \c_zero_dim }
8734 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8735 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int\_step\_inline:nnn is provided by currying.

```

8736 \bool_if:NTF \l_@@_submatrix_slim_bool
8737 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8738 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8739 {
8740     \cs_if_exist:cT
8741     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8742     {
8743         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8744         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8745         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8746     }
8747     \cs_if_exist:cT
8748     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8749     {
8750         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8751         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8752         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8753     }
8754 }
8755 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8756 { \@@_error:nn { Impossible~delimiter } { left } }
8757 {
8758     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8759     { \@@_error:nn { Impossible~delimiter } { right } }
8760     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8761 }
8762 \endpgfpicture
8763 }
8764 \group_end:
8765 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.



```

8766 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8767 {
8768   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8769   \dim_set:Nn \l_@@_y_initial_dim
8770   {
8771     \fp_to_dim:n
8772     {
8773       \pgf@y
8774       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8775     }
8776   }
8777   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8778   \dim_set:Nn \l_@@_y_final_dim
8779   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8780   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8781   {
8782     \cs_if_exist:cT
8783     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8784     {
8785       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8786       \dim_set:Nn \l_@@_y_initial_dim
8787       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8788     }
8789     \cs_if_exist:cT
8790     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8791     {
8792       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8793       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8794       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8795     }
8796   }
8797   \dim_set:Nn \l_tmpa_dim
8798   {
8799     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8800     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8801   }
8802   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8803   \group_begin:
8804   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8805   \@@_set_CT@arc@:o \l_@@_rules_color_tl
8806   \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`.

```

8807   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8808   {
8809     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8810     {
8811       \int_compare:nNnT
8812       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8813       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8814         \@@_qpoint:n { col - ##1 }
8815         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8816         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8817         \pgfusepathqstroke
8818       }
8819     }
8820   }

```

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.

```

8821 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8822 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8823 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8824 {
8825   \bool_lazy_and:nnTF
8826   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8827   {
8828     \int_compare_p:nNn
8829     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8830   {
8831     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8832     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8833     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8834     \pgfusepathqstroke
8835   }
8836   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8837 }

```

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.

```

8838 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8839 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8840 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8841 {
8842   \bool_lazy_and:nnTF
8843   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8844   {
8845     \int_compare_p:nNn
8846     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8847   {
8848     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8849 \group_begin:

```

We compute in `\l_tmpa_dim` the  $x$ -value of the left end of the rule.

```

8850 \dim_set:Nn \l_tmpa_dim
8851 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8852 \str_case:nn { #1 }
8853 {
8854   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8855   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8856   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8857   }
8858   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the  $x$ -value of the right end of the rule.

```

8859 \dim_set:Nn \l_tmpb_dim
8860 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8861 \str_case:nn { #2 }
8862 {
8863   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8864   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8865   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8866 }
8867 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8868 \pgfusepathqstroke
8869 \group_end:
8870 }
8871 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8872 }

```

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).

```

8873 \str_if_empty:NF \l_@@_submatrix_name_str
8874 {
8875   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8876   \l_@@_x_initial_dim \l_@@_y_initial_dim
8877   \l_@@_x_final_dim \l_@@_y_final_dim
8878 }
8879 \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`.

```

8880 \begin { pgfscope }
8881 \pgftransformshift
8882 {
8883   \pgfpoint
8884   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8885   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8886 }
8887 \str_if_empty:NTF \l_@@_submatrix_name_str
8888 { \@@_node_left:nn #1 { } }
8889 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8890 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8891 \pgftransformshift
8892 {
8893   \pgfpoint
8894   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8895   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8896 }
8897 \str_if_empty:NTF \l_@@_submatrix_name_str
8898 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8899 {
8900   \@@_node_right:nnnn #2
8901   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8902 }

```

Now, we deal with the key code of `\SubMatrix`. That key should contain a TikZ instruction and the nodes in that instruction will be relative to the current `\SubMatrix`. That's why we need a redefinition of `\pgfpointanchor`.

```

8903 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8904 \flag_clear_new:N \l_@@_code_flag
8905 \l_@@_code_tl
8906 }

```

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*, *row-i*, *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`.

```

8907 \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.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
8908 \cs_new:Npn \@@_pgfpointanchor:n #1
8909 { \exp_args:Ne \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`).

```
8910 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8911 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }

8912 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8913 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8914 \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8915 { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8916 { \@@_pgfpointanchor_ii:n { #1 } }
8917 }
```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```
8918 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8919 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
8920 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1-\q_stop }
```

```
8921 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2\q_stop
8922 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8923 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8924 { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
8925 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8926 }
```

The following function is for the case when the name contains an hyphen.

```
8927 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8928 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8929 \@@_env:
8930 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8931 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8932 }
```

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.

```

8933 \tl_const:Nn \c_@@_integers_alist_tl
8934 {
8935   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8936   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8937   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8938   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8939 }

8940 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8941 {

```

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  $i$ - $|j$ . That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises with its command `\name_of_command` (see above).

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`.

```

8942   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8943   {
8944     \flag_raise:N \l_@@_code_flag

```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8945     \@@_env: -
8946     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8947       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8948       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8949   }
8950   {
8951     \str_if_eq:eeTF { #1 } { last }
8952     {
8953       \flag_raise:N \l_@@_code_flag
8954       \@@_env: -
8955       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8956         { \int_eval:n { \l_@@_last_i_tl + 1 } }
8957         { \int_eval:n { \l_@@_last_j_tl + 1 } }
8958     }
8959     { #1 }
8960   }
8961 }

```

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`).

```

8962 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8963 {
8964   \pgfnode
8965     { rectangle }
8966     { east }
8967     {
8968       \nullfont
8969       \c_math_toggle_token
8970       \@@_color:o \l_@@_delimiters_color_tl
8971       \left #1
8972       \vcenter
8973       {
8974         \nullfont
8975         \hrule \@height \l_tmpa_dim
8976         \@depth \c_zero_dim

```

```

8977         \@width \c_zero_dim
8978     }
8979     \right .
8980     \c_math_toggle_token
8981 }
8982 { #2 }
8983 { }
8984 }

```

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.

```

8985 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8986 {
8987     \pgfnode
8988     { rectangle }
8989     { west }
8990     {
8991         \nullfont
8992         \c_math_toggle_token
8993         \colorlet { current-color } { . }
8994         \@@_color:o \l_@@_delimiters_color_tl
8995         \left .
8996         \vcenter
8997         {
8998             \nullfont
8999             \hrule \@height \l_tmpa_dim
9000                 \@depth \c_zero_dim
9001                 \@width \c_zero_dim
9002         }
9003         \right #1
9004         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9005         ^ { \color { current-color } \smash { #4 } }
9006         \c_math_toggle_token
9007     }
9008     { #2 }
9009     { }
9010 }

```

## 34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9011 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9012 {
9013     \peek_remove_spaces:n
9014     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
9015 }
9016 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9017 {
9018     \peek_remove_spaces:n
9019     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
9020 }
9021 \keys_define:nn { nicematrix / Brace }
9022 {
9023     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9024     left-shorten .default:n = true ,
9025     left-shorten .value_forbidden:n = true ,

```

```

9026 right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9027 right-shorten .default:n = true ,
9028 right-shorten .value_forbidden:n = true ,
9029 shorten .meta:n = { left-shorten , right-shorten } ,
9030 shorten .value_forbidden:n = true ,
9031 yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9032 yshift .value_required:n = true ,
9033 yshift .initial:n = \c_zero_dim ,
9034 color .tl_set:N = \l_tmpa_tl ,
9035 color .value_required:n = true ,
9036 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9037 }

```

#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.

```

9038 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9039 {
9040   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9041   \@@_compute_i_j:nn { #1 } { #2 }
9042   \bool_lazy_or:nnTF
9043     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9044     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9045     {
9046       \str_if_eq:eeTF { #5 } { under }
9047       { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9048       { \@@_error:nn { Construct~too~large } { \OverBrace } }
9049     }
9050     {
9051       \tl_clear:N \l_tmpa_tl
9052       \keys_set:nn { nicematrix / Brace } { #4 }
9053       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9054       \pgfpicture
9055       \pgfrememberpicturepositiononpagetrue
9056       \pgf@relevantforpicturesizefalse
9057       \bool_if:NT \l_@@_brace_left_shorten_bool
9058       {
9059         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9060         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9061         {
9062           \cs_if_exist:cT
9063             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9064             {
9065               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9066
9067               \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9068               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9069             }
9070         }
9071       }
9072       \bool_lazy_or:nnT
9073         { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9074         { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9075         {
9076           \@@_qpoint:n { col - \l_@@_first_j_tl }
9077           \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9078         }
9079       \bool_if:NT \l_@@_brace_right_shorten_bool
9080       {
9081         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9082         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9083         {

```

```

9084         \cs_if_exist:cT
9085         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9086         {
9087             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9088             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9089             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9090         }
9091     }
9092 }
9093 \bool_lazy_or:nnT
9094 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9095 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9096 {
9097     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9098     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9099 }
9100 \pgfset { inner~sep = \c_zero_dim }
9101 \str_if_eq:eeTF { #5 } { under }
9102 { \@@_underbrace_i:n { #3 } }
9103 { \@@_overbrace_i:n { #3 } }
9104 \endpgfpicture
9105 }
9106 \group_end:
9107 }

```

The argument is the text to put above the brace.

```

9108 \cs_new_protected:Npn \@@_overbrace_i:n #1
9109 {
9110     \@@_qpoint:n { row - \l_@@_first_i_tl }
9111     \pgftransformshift
9112     {
9113         \pgfpoint
9114         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9115         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9116     }
9117     \pgfnode
9118     { rectangle }
9119     { south }
9120     {
9121         \vtop
9122         {
9123             \group_begin:
9124             \everycr { }
9125             \halign
9126             {
9127                 \hfil ## \hfil \crcr
9128                 \bool_if:NTF \l_@@_tabular_bool
9129                 { \begin { tabular } { c } #1 \end { tabular } }
9130                 { $ \begin { array } { c } #1 \end { array } $ }
9131                 \cr
9132                 \c_math_toggle_token
9133                 \overbrace
9134                 {
9135                     \hbox_to_wd:nn
9136                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9137                     { }
9138                 }
9139                 \c_math_toggle_token
9140                 \cr
9141             }
9142             \group_end:
9143         }
9144     }
9145 }

```



```

9146     { }
9147 }

```

The argument is the text to put under the brace.

```

9148 \cs_new_protected:Npn \@@_underbrace_i:n #1
9149 {
9150   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9151   \pgftransformshift
9152   {
9153     \pgfpoint
9154     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9155     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9156   }
9157   \pgfnode
9158   { rectangle }
9159   { north }
9160   {
9161     \group_begin:
9162     \everycr { }
9163     \vbox
9164     {
9165       \halign
9166       {
9167         \hfil ## \hfil \crcr
9168         \c_math_toggle_token
9169         \underbrace
9170         {
9171           \hbox_to_wd:nn
9172           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9173           { }
9174         }
9175         \c_math_toggle_token
9176         \cr
9177         \bool_if:NTF \l_@@_tabular_bool
9178         { \begin { tabular } { c } #1 \end { tabular } }
9179         { $ \begin { array } { c } #1 \end { array } $ }
9180         \cr
9181       }
9182     }
9183     \group_end:
9184   }
9185   { }
9186   { }
9187 }

```

## 35 The commands HBrace et VBrace

```

9188 \hook_gput_code:nnn { begindocument } { . }
9189 {
9190   \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9191   {
9192     \tikzset
9193     {
9194       nicematrix / brace / .style =
9195       {
9196         decoration = { brace , raise = -0.15 em } ,
9197         decorate ,
9198       } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9199         nicematrix / mirrored-brace / .style =
9200         {
9201             nicematrix / brace ,
9202             decoration = mirror ,
9203         }
9204     }
9205 }
9206 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9207 \keys_define:nn { nicematrix / Hbrace }
9208 {
9209     color .code:n = ,
9210     horizontal-labels .code:n = ,
9211     shorten .code:n = ,
9212     shorten-start .code:n = ,
9213     shorten-end .code:n = ,
9214     unknown .code:n = \@_error:n { Unknown~key~for~Hbrace }
9215 }

```

Here we need an “fully expandable” command.

```

9216 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9217 {
9218     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9219     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9220     { \@@_error:n { Hbrace~not~allowed } }
9221 }

```

The following command must *not* be protected.

```

9222 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9223 {
9224     \int_compare:nNnTF \c@iRow < 1
9225     {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9226     \str_if_eq:nnTF { #2 } { * }
9227     {
9228         \NiceMatrixOptions{nullify-dots}
9229         \Ldots
9230         [
9231             line-style = nicematrix / brace ,
9232             #1 ,
9233             up =
9234             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9235         ]
9236     }
9237     {
9238         \Hdotsfor
9239         [
9240             line-style = nicematrix / brace ,
9241             #1 ,
9242             up =
9243             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9244         ]
9245         { #2 }
9246     }
9247 }
9248 {
9249     \str_if_eq:nnTF { #2 } { * }
9250     {
9251         \NiceMatrixOptions{nullify-dots}

```

```

9252         \Ldots
9253         [
9254             line-style = nicematrix / mirrored-brace ,
9255             #1 ,
9256             down =
9257                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9258         ]
9259     }
9260     {
9261         \Hdotsfor
9262         [
9263             line-style = nicematrix / mirrored-brace ,
9264             #1 ,
9265             down =
9266                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9267         ]
9268         { #2 }
9269     }
9270 }
9271 \keys_set:nn { nicematrix / Hbrace } { #1 }
9272 }

```

Here we need an “fully expandable” command.

```

9273 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9274 {
9275     \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9276     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9277     { \@@_error:n { Vbrace~not~allowed } }
9278 }

```

The following command must *not* be protected.

```

9279 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9280 {
9281     \int_compare:nNnTF \c@jCol = 0
9282     {
9283         \str_if_eq:nnTF { #2 } { * }
9284         {
9285             \NiceMatrixOptions{nullify-dots}
9286             \Vdots
9287             [
9288                 line-style = nicematrix / mirrored-brace ,
9289                 #1 ,
9290                 down =
9291                     \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9292             ]
9293         }
9294     }
9295     \Vdotsfor
9296     [
9297         line-style = nicematrix / mirrored-brace ,
9298         #1 ,
9299         down =
9300             \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9301     ]
9302     { #2 }
9303 }
9304 }
9305 {
9306     \str_if_eq:nnTF { #2 } { * }
9307     {
9308         \NiceMatrixOptions{nullify-dots}
9309         \Vdots
9310         [
9311             line-style = nicematrix / brace ,

```

```

9312         #1 ,
9313         up =
9314         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9315     ]
9316 }
9317 {
9318     \Vdotsfor
9319     [
9320         line-style = nicematrix / brace ,
9321         #1 ,
9322         up =
9323         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9324     ]
9325     { #2 }
9326 }
9327 }
9328 \keys_set:nn { nicematrix / Hbrace } { #1 }
9329 }

```

## 36 The command TikzEveryCell

```

9330 \bool_new:N \l_@@_not_empty_bool
9331 \bool_new:N \l_@@_empty_bool
9332
9333 \keys_define:nn { nicematrix / TikzEveryCell }
9334 {
9335     not-empty .code:n =
9336         \bool_lazy_or:nnTF
9337         \l_@@_in_code_after_bool
9338         \g_@@_recreate_cell_nodes_bool
9339         { \bool_set_true:N \l_@@_not_empty_bool }
9340         { \@@_error:n { detection-of-empty-cells } } ,
9341     not-empty .value_forbidden:n = true ,
9342     empty .code:n =
9343         \bool_lazy_or:nnTF
9344         \l_@@_in_code_after_bool
9345         \g_@@_recreate_cell_nodes_bool
9346         { \bool_set_true:N \l_@@_empty_bool }
9347         { \@@_error:n { detection-of-empty-cells } } ,
9348     empty .value_forbidden:n = true ,
9349     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9350 }
9351
9352
9353 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9354 {
9355     \IfPackageLoadedTF { tikz }
9356     {
9357         \group_begin:
9358         \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.

```

9359         \tl_set:Nn \l_tmpa_tl { { #2 } }
9360         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9361         { \@@_for_a_block:nnnnn ##1 }
9362         \@@_all_the_cells:
9363         \group_end:
9364     }
9365     { \@@_error:n { TikzEveryCell-without-tikz } }
9366 }
9367

```

```

9368 \tl_new:N \@@_i_tl
9369 \tl_new:N \@@_j_tl
9370
9371
9372 \cs_new_protected:Nn \@@_all_the_cells:
9373 {
9374   \int_step_variable:nNn \c@iRow \@@_i_tl
9375   {
9376     \int_step_variable:nNn \c@jCol \@@_j_tl
9377     {
9378       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9379       {
9380         \clist_if_in:NcF \l_@@_corners_cells_clist
9381         { \@@_i_tl - \@@_j_tl }
9382         {
9383           \bool_set_false:N \l_tmpa_bool
9384           \cs_if_exist:cTF
9385           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9386           {
9387             \bool_if:NF \l_@@_empty_bool
9388             { \bool_set_true:N \l_tmpa_bool }
9389           }
9390           {
9391             \bool_if:NF \l_@@_not_empty_bool
9392             { \bool_set_true:N \l_tmpa_bool }
9393           }
9394           \bool_if:NT \l_tmpa_bool
9395           {
9396             \@@_block_tikz:onnnn
9397             \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9398           }
9399         }
9400       }
9401     }
9402   }
9403 }
9404
9405 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9406 {
9407   \bool_if:NF \l_@@_empty_bool
9408   {
9409     \@@_block_tikz:onnnn
9410     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9411   }
9412   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9413 }
9414
9415 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9416 {
9417   \int_step_inline:nnn { #1 } { #3 }
9418   {
9419     \int_step_inline:nnn { #2 } { #4 }
9420     { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9421   }
9422 }

```

## 37 The command \ShowCellNames

```

9423 \NewDocumentCommand \@@_ShowCellNames { }
9424 {
9425   \bool_if:NT \l_@@_in_code_after_bool
9426   {

```

```

9427 \pgfpicture
9428 \pgfrememberpicturerepositiononpagetrue
9429 \pgf@relevantforpicturesizefalse
9430 \pgfpathrectanglecorners
9431 { \@@_qpoint:n { 1 } }
9432 {
9433 \@@_qpoint:n
9434 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9435 }
9436 \pgfsetfillopacity { 0.75 }
9437 \pgfsetfillcolor { white }
9438 \pgfusepathqfill
9439 \endpgfpicture
9440 }
9441 \dim_gzero_new:N \g_@@_tmpc_dim
9442 \dim_gzero_new:N \g_@@_tmpd_dim
9443 \dim_gzero_new:N \g_@@_tmpe_dim
9444 \int_step_inline:nn \c@iRow
9445 {
9446 \bool_if:NTF \l_@@_in_code_after_bool
9447 {
9448 \pgfpicture
9449 \pgfrememberpicturerepositiononpagetrue
9450 \pgf@relevantforpicturesizefalse
9451 }
9452 { \begin { pgfpicture } }
9453 \@@_qpoint:n { row - ##1 }
9454 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9455 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9456 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9457 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9458 \bool_if:NTF \l_@@_in_code_after_bool
9459 { \endpgfpicture }
9460 { \end { pgfpicture } }
9461 \int_step_inline:nn \c@jCol
9462 {
9463 \hbox_set:Nn \l_tmpa_box
9464 {
9465 \normalfont \Large \sffamily \bfseries
9466 \bool_if:NTF \l_@@_in_code_after_bool
9467 { \color { red } }
9468 { \color { red ! 50 } }
9469 ##1 - ####1
9470 }
9471 \bool_if:NTF \l_@@_in_code_after_bool
9472 {
9473 \pgfpicture
9474 \pgfrememberpicturerepositiononpagetrue
9475 \pgf@relevantforpicturesizefalse
9476 }
9477 { \begin { pgfpicture } }
9478 \@@_qpoint:n { col - ####1 }
9479 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9480 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9481 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9482 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9483 \bool_if:NTF \l_@@_in_code_after_bool
9484 { \endpgfpicture }
9485 { \end { pgfpicture } }
9486 \fp_set:Nn \l_tmpa_fp
9487 {
9488 \fp_min:nn
9489 {

```

```

9490         \fp_min:nn
9491         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9492         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9493     }
9494     { 1.0 }
9495 }
9496 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9497 \pgfpicture
9498 \pgfrememberpicturepositiononpagetrue
9499 \pgf@relevantforpicturesizefalse
9500 \pgftransformshift
9501 {
9502     \pgfpoint
9503     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9504     { \dim_use:N \g_tmpa_dim }
9505 }
9506 \pgfnode
9507 { rectangle }
9508 { center }
9509 { \box_use:N \l_tmpa_box }
9510 { }
9511 { }
9512 \endpgfpicture
9513 }
9514 }
9515 }

```

## 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.

```

9516 \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.

```

9517 \bool_new:N \g_@@_footnote_bool
9518 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9519 {
9520     You~have~used~the~key~'\l_keys_key_str'~when~loading~nicematrix~
9521     but~that~key~is~unknown. \\
9522     It~will~be~ignored. \\
9523     For~a~list~of~the~available~keys,~type~H~<return>.
9524 }
9525 {
9526     The~available~keys~are~(in~alphabetic~order):~
9527     footnote,~
9528     footnotehyper,~
9529     messages~for~Overleaf,~
9530     renew~dots~and~
9531     renew~matrix.
9532 }
9533 \keys_define:nn { nicematrix }
9534 {
9535     renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9536     renew~dots .value_forbidden:n = true ,

```

```

9537     renew-matrix .code:n = \@@_renew_matrix: ,
9538     renew-matrix .value_forbidden:n = true ,
9539     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9540     footnote .bool_set:N = \g_@@_footnote_bool ,
9541     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9542     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9543   }
9544   \ProcessKeyOptions

9545   \@@_msg_new:nn { footnote-with-footnotehyper-package }
9546   {
9547     You~can't~use~the~option~'footnote'~because~the~package~
9548     footnotehyper~has~already~been~loaded.~
9549     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9550     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9551     of~the~package~footnotehyper.\\
9552     The~package~footnote~won't~be~loaded.
9553   }

9554   \@@_msg_new:nn { footnotehyper-with-footnote-package }
9555   {
9556     You~can't~use~the~option~'footnotehyper'~because~the~package~
9557     footnote~has~already~been~loaded.~
9558     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9559     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9560     of~the~package~footnote.\\
9561     The~package~footnotehyper~won't~be~loaded.
9562   }

9563   \bool_if:NT \g_@@_footnote_bool
9564   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9565     \IfClassLoadedTF { beamer }
9566     { \bool_set_false:N \g_@@_footnote_bool }
9567     {
9568       \IfPackageLoadedTF { footnotehyper }
9569       { \@@_error:n { footnote-with-footnotehyper-package } }
9570       { \usepackage { footnote } }
9571     }
9572   }

9573   \bool_if:NT \g_@@_footnotehyper_bool
9574   {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9575     \IfClassLoadedTF { beamer }
9576     { \bool_set_false:N \g_@@_footnote_bool }
9577     {
9578       \IfPackageLoadedTF { footnote }
9579       { \@@_error:n { footnotehyper-with-footnote-package } }
9580       { \usepackage { footnotehyper } }
9581     }
9582     \bool_set_true:N \g_@@_footnote_bool
9583   }

```

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.

```
9584 \bool_new:N \l_@@_underscore_loaded_bool
9585 \IfPackageLoadedT { underscore }
9586 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9587 \hook_gput_code:nnn { begindocument } { . }
9588 {
9589   \bool_if:NF \l_@@_underscore_loaded_bool
9590   {
9591     \IfPackageLoadedT { underscore }
9592     { \@@_error:n { underscore-after-nicematrix } }
9593   }
9594 }
```

## 40 Error messages of the package

```
9595 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9596 { \str_const:Nn \c_@@_available_keys_str { } }
9597 {
9598   \str_const:Nn \c_@@_available_keys_str
9599   { For-a-list-of-the-available-keys,-type-H<return>. }
9600 }
9601 \seq_new:N \g_@@_types_of_matrix_seq
9602 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9603 {
9604   NiceMatrix ,
9605   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9606 }
9607 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9608 { \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:NoF` 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`.

```
9609 \cs_new_protected:Npn \@@_error_too_much_cols:
9610 {
9611   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9612   { \@@_fatal:nn { too-much-cols-for-array } }
9613   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9614   { \@@_fatal:n { too-much-cols-for-matrix } }
9615   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9616   { \@@_fatal:n { too-much-cols-for-matrix } }
9617   \bool_if:NF \l_@@_last_col_without_value_bool
9618   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9619 }
```

The following command must *not* be protected since it's used in an error message.

```
9620 \cs_new:Npn \@@_message_hdotsfor:
9621 {
9622   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9623   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9624 }
9625 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
```

```

9626 {
9627   Incompatible~options.\\
9628   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9629   The~output~will~not~be~reliable.
9630 }
9631 \\@@_msg_new:nn { key~color~inside }
9632 {
9633   Key~deprecated.\\
9634   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9635   and~have~been~deprecated.\\
9636   You~won't~have~similar~message~till~the~end~of~the~document.
9637 }
9638 \\@@_msg_new:nn { negative~weight }
9639 {
9640   Negative~weight.\\
9641   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9642   the~value~'\int_use:N \l_@@_weight_int'.\\
9643   The~absolute~value~will~be~used.
9644 }
9645 \\@@_msg_new:nn { last~col~not~used }
9646 {
9647   Column~not~used.\\
9648   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9649   in~your~\\@@_full_name_env:~.~However,~you~can~go~on.
9650 }
9651 \\@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9652 {
9653   Too~much~columns.\\
9654   In~the~row~\int_eval:n { \c@iRow },~
9655   you~try~to~use~more~columns~
9656   than~allowed~by~your~\\@@_full_name_env:~.\\@@_message_hdotsfor:\\
9657   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9658   (plus~the~exterior~columns).~This~error~is~fatal.
9659 }
9660 \\@@_msg_new:nn { too~much~cols~for~matrix }
9661 {
9662   Too~much~columns.\\
9663   In~the~row~\int_eval:n { \c@iRow },~
9664   you~try~to~use~more~columns~than~allowed~by~your~
9665   \\@@_full_name_env:~.\\@@_message_hdotsfor:\\ Recall~that~the~maximal~
9666   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9667   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9668   Its~current~value~is~\int_use:N \c@MaxMatrixCols\\ (use~
9669   \token_to_str:N \setcounter\\ to~change~that~value).~
9670   This~error~is~fatal.
9671 }
9672 \\@@_msg_new:nn { too~much~cols~for~array }
9673 {
9674   Too~much~columns.\\
9675   In~the~row~\int_eval:n { \c@iRow },~
9676   ~you~try~to~use~more~columns~than~allowed~by~your~
9677   \\@@_full_name_env:~.\\@@_message_hdotsfor:\\ The~maximal~number~of~columns~is~
9678   \int_use:N \g_@@_static_num_of_col_int\\
9679   \bool_if:nT
9680     { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9681     { ~(plus~the~exterior~ones) }
9682   since~the~preamble~is~'\g_@@_user_preamble_tl'.\\
9683   This~error~is~fatal.
9684 }
9685 \\@@_msg_new:nn { columns~not~used }

```

```

9686 {
9687   Columns~not~used.\\
9688   The~preamble~of~your~\@@_full_name_env:\ is~'\g_@@_user_preamble_tl'.~
9689   It~announces~\int_use:N
9690   \g_@@_static_num_of_col_int\ columns~but~you~only~used~\int_use:N \c@jCol.\\
9691   The~columns~you~did~not~used~won't~be~created.\\
9692   You~won't~have~similar~warning~till~the~end~of~the~document.
9693 }
9694 \@@_msg_new:nn { empty~preamble }
9695 {
9696   Empty~preamble.\\
9697   The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9698   This~error~is~fatal.
9699 }
9700 \@@_msg_new:nn { in~first~col }
9701 {
9702   Erroneous~use.\\
9703   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9704   That~command~will~be~ignored.
9705 }
9706 \@@_msg_new:nn { in~last~col }
9707 {
9708   Erroneous~use.\\
9709   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9710   That~command~will~be~ignored.
9711 }
9712 \@@_msg_new:nn { in~first~row }
9713 {
9714   Erroneous~use.\\
9715   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9716   That~command~will~be~ignored.
9717 }
9718 \@@_msg_new:nn { in~last~row }
9719 {
9720   Erroneous~use.\\
9721   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9722   That~command~will~be~ignored.
9723 }
9724 \@@_msg_new:nn { TopRule~without~booktabs }
9725 {
9726   Erroneous~use.\\
9727   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9728   That~command~will~be~ignored.
9729 }
9730 \@@_msg_new:nn { TopRule~without~tikz }
9731 {
9732   Erroneous~use.\\
9733   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9734   That~command~will~be~ignored.
9735 }
9736 \@@_msg_new:nn { caption~outside~float }
9737 {
9738   Key~caption~forbidden.\\
9739   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9740   environment.~This~key~will~be~ignored.
9741 }
9742 \@@_msg_new:nn { short~caption~without~caption }
9743 {
9744   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9745   However,~your~'short~caption'~will~be~used~as~'caption'.

```

```

9746 }
9747 \@@_msg_new:nn { double~closing~delimiter }
9748 {
9749   Double~delimiter.\\
9750   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9751   delimiter.~This~delimiter~will~be~ignored.
9752 }
9753 \@@_msg_new:nn { delimiter~after~opening }
9754 {
9755   Double~delimiter.\\
9756   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9757   delimiter.~That~delimiter~will~be~ignored.
9758 }
9759 \@@_msg_new:nn { bad~option~for~line~style }
9760 {
9761   Bad~line~style.\\
9762   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9763   is~'standard'.~That~key~will~be~ignored.
9764 }
9765 \@@_msg_new:nn { corners~with~no~cell~nodes }
9766 {
9767   Incompatible~keys.\\
9768   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9769   is~in~force.\\
9770   If~you~go~on,~that~key~will~be~ignored.
9771 }
9772 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9773 {
9774   Incompatible~keys.\\
9775   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9776   is~in~force.\\
9777   If~you~go~on,~those~extra~nodes~won't~be~created.
9778 }
9779 \@@_msg_new:nn { Identical~notes~in~caption }
9780 {
9781   Identical~tabular~notes.\\
9782   You~can't~put~several~notes~with~the~same~content~in~
9783   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9784   If~you~go~on,~the~output~will~probably~be~erroneous.
9785 }
9786 \@@_msg_new:nn { tabularnote~below~the~tabular }
9787 {
9788   \token_to_str:N \tabularnote\ forbidden\\
9789   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9790   of~your~tabular~because~the~caption~will~be~composed~below~
9791   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9792   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9793   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9794   no~similar~error~will~raised~in~this~document.
9795 }
9796 \@@_msg_new:nn { Unknown~key~for~rules }
9797 {
9798   Unknown~key.\\
9799   There~is~only~two~keys~available~here:~width~and~color.\\
9800   Your~key~'\l_keys_key_str'~will~be~ignored.
9801 }
9802 \@@_msg_new:nn { Unknown~key~for~Hbrace }
9803 {
9804   Unknown~key.\\
9805   You~have~used~the~key~'\l_keys_key_str'~but~the~only~

```

```

9806     keys-allowed-for-the-commands~\token_to_str:N \Hbrace\
9807     and~\token_to_str:N \Vbrace\ are:~'color',~
9808     'horizontal-labels',~'shorten'~'shorten-end'~
9809     and~'shorten-start'.
9810 }
9811 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9812 {
9813     Unknown~key.\\
9814     There~is~only~two~keys~available~here:~
9815     'empty'~and~'not-empty'.\\
9816     Your~key~'\l_keys_key_str'~will~be~ignored.
9817 }
9818 \@@_msg_new:nn { Unknown~key~for~rotate }
9819 {
9820     Unknown~key.\\
9821     The~only~key~available~here~is~'c'.\\
9822     Your~key~'\l_keys_key_str'~will~be~ignored.
9823 }
9824 \@@_msg_new:nnn { Unknown~key~for~custom-line }
9825 {
9826     Unknown~key.\\
9827     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
9828     It~you~go~on,~you~will~probably~have~other~errors. \\
9829     \c_@@_available_keys_str
9830 }
9831 {
9832     The~available~keys~are~(in~alphabetic~order):~
9833     ccommand,~
9834     color,~
9835     command,~
9836     dotted,~
9837     letter,~
9838     multiplicity,~
9839     sep-color,~
9840     tikz,~and~total-width.
9841 }
9842 \@@_msg_new:nnn { Unknown~key~for~xdots }
9843 {
9844     Unknown~key.\\
9845     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9846     \c_@@_available_keys_str
9847 }
9848 {
9849     The~available~keys~are~(in~alphabetic~order):~
9850     'color',~
9851     'horizontal-labels',~
9852     'inter',~
9853     'line-style',~
9854     'radius',~
9855     'shorten',~
9856     'shorten-end'~and~'shorten-start'.
9857 }
9858 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9859 {
9860     Unknown~key.\\
9861     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9862     (and~you~try~to~use~'\l_keys_key_str')\\
9863     That~key~will~be~ignored.
9864 }
9865 \@@_msg_new:nn { label~without~caption }
9866 {

```

```

9867     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9868     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9869 }

9870 \@@_msg_new:nn { W~warning }
9871 {
9872     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9873     (row~\int_use:N \c@iRow).
9874 }

9875 \@@_msg_new:nn { Construct~too~large }
9876 {
9877     Construct~too~large.\\
9878     Your~command~\token_to_str:N #1
9879     can't~be~drawn~because~your~matrix~is~too~small.\\
9880     That~command~will~be~ignored.
9881 }

9882 \@@_msg_new:nn { underscore~after~nicematrix }
9883 {
9884     Problem~with~'underscore'.\\
9885     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9886     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9887     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9888 }

9889 \@@_msg_new:nn { ampersand~in~light~syntax }
9890 {
9891     Ampersand~forbidden.\\
9892     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9893     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9894 }

9895 \@@_msg_new:nn { double~backslash~in~light~syntax }
9896 {
9897     Double~backslash~forbidden.\\
9898     You~can't~use~\token_to_str:N
9899     \\~to~separate~rows~because~the~key~'light-syntax'~
9900     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9901     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9902 }

9903 \@@_msg_new:nn { hlines~with~color }
9904 {
9905     Incompatible~keys.\\
9906     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9907     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9908     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9909     Your~key~will~be~discarded.
9910 }

9911 \@@_msg_new:nn { bad~value~for~baseline }
9912 {
9913     Bad~value~for~baseline.\\
9914     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9915     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9916     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9917     the~form~'line-i'.\\
9918     A~value~of~1~will~be~used.
9919 }

9920 \@@_msg_new:nn { detection~of~empty~cells }
9921 {
9922     Problem~with~'not-empty'\\
9923     For~technical~reasons,~you~must~activate~
9924     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9925     in~order~to~use~the~key~'\l_keys_key_str'.\\
9926     That~key~will~be~ignored.

```

```

9927 }
9928 \@@_msg_new:nn { siunitx~not~loaded }
9929 {
9930   siunitx~not~loaded\\
9931   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9932   That~error~is~fatal.
9933 }
9934 \@@_msg_new:nn { Invalid~name }
9935 {
9936   Invalid~name.\\
9937   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9938   \SubMatrix\ of~your~\@@_full_name_env:.\\
9939   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9940   This~key~will~be~ignored.
9941 }
9942 \@@_msg_new:nn { Hbrace~not~allowed }
9943 {
9944   Command~not~allowed.\\
9945   You~can't~use~the~command~\token_to_str:N \Hbrace\
9946   because~you~have~not~loaded~TikZ~
9947   and~the~TikZ~library~'decorations.pathreplacing'.\\
9948   Use:~\token_to_str:N \usepackage{tikz}~
9949   \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9950   That~command~will~be~ignored.
9951 }
9952 \@@_msg_new:nn { Vbrace~not~allowed }
9953 {
9954   Command~not~allowed.\\
9955   You~can't~use~the~command~\token_to_str:N \Vbrace\
9956   because~you~have~not~loaded~TikZ~
9957   and~the~TikZ~library~'decorations.pathreplacing'.\\
9958   Use:~\token_to_str:N \usepackage{tikz}~
9959   \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9960   That~command~will~be~ignored.
9961 }
9962 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9963 {
9964   Wrong~line.\\
9965   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9966   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9967   number~is~not~valid.~It~will~be~ignored.
9968 }
9969 \@@_msg_new:nn { Impossible~delimiter }
9970 {
9971   Impossible~delimiter.\\
9972   It's~impossible~to~draw~the~#1~delimiter~of~your~
9973   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9974   in~that~column.
9975   \bool_if:NT \l_@@_submatrix_slim_bool
9976   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9977   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9978 }
9979 \@@_msg_new:nnn { width~without~X~columns }
9980 {
9981   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
9982   the~preamble~('g_@@_user_preamble_tl')~of~your~\@@_full_name_env:.\\
9983   That~key~will~be~ignored.
9984 }
9985 {
9986   This~message~is~the~message~'width~without~X~columns'~
9987   of~the~module~'nicematrix'.~

```

```

9988     The~experimented~users~can~disable~that~message~with~
9989     \token_to_str:N \msg_redirect_name:nnn.\\
9990   }
9991
9992   \@@_msg_new:nn { key~multiplicity~with~dotted }
9993   {
9994     Incompatible~keys. \\
9995     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9996     in~a~'custom~line'.~They~are~incompatible. \\
9997     The~key~'multiplicity'~will~be~discarded.
9998   }
9999   \@@_msg_new:nn { empty~environment }
10000   {
10001     Empty~environment.\\
10002     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
10003   }
10004   \@@_msg_new:nn { No~letter~and~no~command }
10005   {
10006     Erroneous~use.\\
10007     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
10008     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10009     '~ccommand'~(to~draw~horizontal~rules).\\
10010     However,~you~can~go~on.
10011   }
10012   \@@_msg_new:nn { Forbidden~letter }
10013   {
10014     Forbidden~letter.\\
10015     You~can't~use~the~letter~'#1'~for~a~customized~line.~
10016     It~will~be~ignored.\\
10017     The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10018   }
10019   \@@_msg_new:nn { Several~letters }
10020   {
10021     Wrong~name.\\
10022     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10023     have~used~'\l_@@_letter_str').\\
10024     It~will~be~ignored.
10025   }
10026   \@@_msg_new:nn { Delimiter~with~small }
10027   {
10028     Delimiter~forbidden.\\
10029     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
10030     because~the~key~'small'~is~in~force.\\
10031     This~error~is~fatal.
10032   }
10033   \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10034   {
10035     Unknown~cell.\\
10036     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
10037     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
10038     can't~be~executed~because~a~cell~doesn't~exist.\\
10039     This~command~\token_to_str:N \line\ will~be~ignored.
10040   }
10041   \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10042   {
10043     Duplicate~name.\\
10044     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
10045     in~this~\@@_full_name_env:.\
10046     This~key~will~be~ignored.\\
10047     \bool_if:NF \g_@@_messages_for_Overleaf_bool

```



```

10048     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10049 }
10050 {
10051     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
10052     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
10053 }
10054 \@@_msg_new:nn { r~or~l~with~preamble }
10055 {
10056     Erroneous~use.\\
10057     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
10058     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10059     your~\@@_full_name_env:~
10060     This~key~will~be~ignored.
10061 }
10062 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10063 {
10064     Erroneous~use.\\
10065     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
10066     the~array.~This~error~is~fatal.
10067 }
10068 \@@_msg_new:nn { bad~corner }
10069 {
10070     Bad~corner.\\
10071     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10072     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10073     This~specification~of~corner~will~be~ignored.
10074 }
10075 \@@_msg_new:nn { bad~border }
10076 {
10077     Bad~border.\\
10078     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
10079     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
10080     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10081     also~use~the~key~'tikz'
10082     \IfPackageLoadedF { tikz }
10083     {~if~you~load~the~LaTeX~package~'tikz'}).\\
10084     This~specification~of~border~will~be~ignored.
10085 }
10086 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10087 {
10088     TikZ~not~loaded.\\
10089     You~can't~use~\token_to_str:N \TikzEveryCell\
10090     because~you~have~not~loaded~tikz.~
10091     This~command~will~be~ignored.
10092 }
10093 \@@_msg_new:nn { tikz~key~without~tikz }
10094 {
10095     TikZ~not~loaded.\\
10096     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
10097     \Block'~because~you~have~not~loaded~tikz.~
10098     This~key~will~be~ignored.
10099 }
10100 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
10101 {
10102     Erroneous~use.\\
10103     In~the~\@@_full_name_env:,~you~must~use~the~key~
10104     'last~col'~without~value.\\
10105     However,~you~can~go~on~for~this~time~
10106     (the~value~'\l_keys_value_tl'~will~be~ignored).
10107 }

```

```

10108 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
10109 {
10110   Erroneous-use.\
10111   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
10112   'last-col'~without~value.\
10113   However,~you~can~go~on~for~this~time~
10114   (the~value~'\l_keys_value_tl'~will~be~ignored).
10115 }
10116 \@@_msg_new:nn { Block-too-large-1 }
10117 {
10118   Block-too-large.\
10119   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10120   too~small~for~that~block. \
10121   This~block~and~maybe~others~will~be~ignored.
10122 }
10123 \@@_msg_new:nn { Block-too-large-2 }
10124 {
10125   Block-too-large.\
10126   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
10127   \g_@@_static_num_of_col_int\
10128   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
10129   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10130   (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
10131   This~block~and~maybe~others~will~be~ignored.
10132 }
10133 \@@_msg_new:nn { unknown-column-type }
10134 {
10135   Bad-column-type.\
10136   The~column~type~'#1'~in~your~\@@_full_name_env:\
10137   is~unknown. \
10138   This~error~is~fatal.
10139 }
10140 \@@_msg_new:nn { unknown-column-type-S }
10141 {
10142   Bad-column-type.\
10143   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \
10144   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10145   load~that~package. \
10146   This~error~is~fatal.
10147 }
10148 \@@_msg_new:nn { tabularnote-forbidden }
10149 {
10150   Forbidden-command.\
10151   You~can't~use~the~command~\token_to_str:N\tabularnote\
10152   ~here.~This~command~is~available~only~in~
10153   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10154   the~argument~of~a~command~\token_to_str:N \caption\ included~
10155   in~an~environment~{table}. \
10156   This~command~will~be~ignored.
10157 }
10158 \@@_msg_new:nn { borders-forbidden }
10159 {
10160   Forbidden-key.\
10161   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
10162   because~the~option~'rounded-corners'~
10163   is~in~force~with~a~non-zero~value.\
10164   This~key~will~be~ignored.
10165 }
10166 \@@_msg_new:nn { bottomrule-without-booktabs }
10167 {
10168   booktabs~not~loaded.\

```

```

10169     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10170     loaded~'booktabs'.\\
10171     This~key~will~be~ignored.
10172 }
10173 \@@_msg_new:nn { enumitem~not~loaded }
10174 {
10175     enumitem~not~loaded.\\
10176     You~can't~use~the~command~\token_to_str:N\tabularnote\
10177     ~because~you~haven't~loaded~'enumitem'.\\
10178     All~the~commands~\token_to_str:N\tabularnote\ will~be~
10179     ignored~in~the~document.
10180 }
10181 \@@_msg_new:nn { tikz~without~tikz }
10182 {
10183     Tikz~not~loaded.\\
10184     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10185     loaded.~If~you~go~on,~that~key~will~be~ignored.
10186 }
10187 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
10188 {
10189     Tikz~not~loaded.\\
10190     You~have~used~the~key~'tikz'~in~the~definition~of~a~
10191     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10192     You~can~go~on~but~you~will~have~another~error~if~you~actually~
10193     use~that~custom~line.
10194 }
10195 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10196 {
10197     Tikz~not~loaded.\\
10198     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10199     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
10200     That~key~will~be~ignored.
10201 }
10202 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10203 {
10204     Erroneous~use.\\
10205     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10206     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10207     The~key~'color'~will~be~discarded.
10208 }
10209 \@@_msg_new:nn { Wrong~last~row }
10210 {
10211     Wrong~number.\\
10212     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
10213     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
10214     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
10215     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10216     without~value~(more~compilations~might~be~necessary).
10217 }
10218 \@@_msg_new:nn { Yet~in~env }
10219 {
10220     Nested~environments.\\
10221     Environments~of~nicematrix~can't~be~nested.\\
10222     This~error~is~fatal.
10223 }
10224 \@@_msg_new:nn { Outside~math~mode }
10225 {
10226     Outside~math~mode.\\
10227     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
10228     (and~not~in~\token_to_str:N \vcenter).\\

```

```

10229     This-error-is-fatal.
10230 }

10231 \@@_msg_new:nn { One-letter-allowed }
10232 {
10233     Bad-name.\
10234     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1~and~
10235     you-have-used~'\l_keys_value_tl'~.\
10236     It~will~be~ignored.
10237 }

10238 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10239 {
10240     Environment~{TabularNote}~forbidden.\
10241     You-must-use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
10242     but~*before*~the~\token_to_str:N \CodeAfter.\
10243     This-environment~{TabularNote}~will~be~ignored.
10244 }

10245 \@@_msg_new:nn { varwidth-not-loaded }
10246 {
10247     varwidth-not-loaded.\
10248     You-can't-use-the-column-type~'V'~because~'varwidth'~is~not~
10249     loaded.\
10250     Your-column~will~behave~like~'p'.
10251 }

10252 \@@_msg_new:nnn { Unknown-key-for-RulesBis }
10253 {
10254     Unknown-key.\
10255     Your-key~'\l_keys_key_str'~is-unknown-for-a-rule.\
10256     \c_@@_available_keys_str
10257 }
10258 {
10259     The-available-keys-are~(in-alphabetic-order):~
10260     color,~
10261     dotted,~
10262     multiplicity,~
10263     sep-color,~
10264     tikz,~and~total-width.
10265 }
10266

10267 \@@_msg_new:nnn { Unknown-key-for-Block }
10268 {
10269     Unknown-key.\
10270     The-key~'\l_keys_key_str'~is-unknown-for-the-command~\token_to_str:N
10271     \Block.\ It~will~be~ignored. \
10272     \c_@@_available_keys_str
10273 }
10274 {
10275     The-available-keys-are~(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
10276     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10277     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10278     and~vlines.
10279 }

10280 \@@_msg_new:nnn { Unknown-key-for-Brace }
10281 {
10282     Unknown-key.\
10283     The-key~'\l_keys_key_str'~is-unknown-for-the-commands~\token_to_str:N
10284     \UnderBrace\ and~\token_to_str:N \OverBrace.\
10285     It~will~be~ignored. \
10286     \c_@@_available_keys_str
10287 }
10288 {
10289     The-available-keys-are~(in-alphabetic-order):~color,~left-shorten,~

```

```

10290     right-shorten,~shorten~(which~fixes~both~left~shorten~and~
10291     right-shorten)~and~yshift.
10292 }

10293 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10294 {
10295     Unknown~key.\\
10296     The~key~'\l_keys_key_str'~is~unknown.\\
10297     It~will~be~ignored. \\
10298     \c_@@_available_keys_str
10299 }
10300 {
10301     The~available~keys~are~(in~alphabetic~order):~
10302     delimiters/color,~
10303     rules~(with~the~subkeys~'color'~and~'width'),~
10304     sub-matrix~(several~subkeys)~
10305     and~xdots~(several~subkeys).~
10306     The~latter~is~for~the~command~\token_to_str:N \line.
10307 }

10308 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10309 {
10310     Unknown~key.\\
10311     The~key~'\l_keys_key_str'~is~unknown.\\
10312     It~will~be~ignored. \\
10313     \c_@@_available_keys_str
10314 }
10315 {
10316     The~available~keys~are~(in~alphabetic~order):~
10317     create-cell-nodes,~
10318     delimiters/color~and~
10319     sub-matrix~(several~subkeys).
10320 }

10321 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10322 {
10323     Unknown~key.\\
10324     The~key~'\l_keys_key_str'~is~unknown.\\
10325     That~key~will~be~ignored. \\
10326     \c_@@_available_keys_str
10327 }
10328 {
10329     The~available~keys~are~(in~alphabetic~order):~
10330     'delimiters/color',~
10331     'extra-height',~
10332     'hlines',~
10333     'hvlines',~
10334     'left-xshift',~
10335     'name',~
10336     'right-xshift',~
10337     'rules'~(with~the~subkeys~'color'~and~'width'),~
10338     'slim',~
10339     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10340     and~'right-xshift').\\
10341 }

10342 \@@_msg_new:nnn { Unknown~key~for~notes }
10343 {
10344     Unknown~key.\\
10345     The~key~'\l_keys_key_str'~is~unknown.\\
10346     That~key~will~be~ignored. \\
10347     \c_@@_available_keys_str
10348 }
10349 {
10350     The~available~keys~are~(in~alphabetic~order):~
10351     bottomrule,~

```

```

10352     code-after,~
10353     code-before,~
10354     detect-duplicates,~
10355     enumitem-keys,~
10356     enumitem-keys-para,~
10357     para,~
10358     label-in-list,~
10359     label-in-tabular~and~
10360     style.
10361 }

10362 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10363 {
10364     Unknown~key.\\
10365     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10366     \token_to_str:N \RowStyle. \\
10367     That~key~will~be~ignored. \\
10368     \c_@@_available_keys_str
10369 }
10370 {
10371     The~available~keys~are~(in~alphabetic~order):~
10372     bold,~
10373     cell-space-top-limit,~
10374     cell-space-bottom-limit,~
10375     cell-space-limits,~
10376     color,~
10377     fill~(alias:~rowcolor),~
10378     nb-rows,
10379     opacity~and~
10380     rounded-corners.
10381 }

10382 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10383 {
10384     Unknown~key.\\
10385     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10386     \token_to_str:N \NiceMatrixOptions. \\
10387     That~key~will~be~ignored. \\
10388     \c_@@_available_keys_str
10389 }
10390 {
10391     The~available~keys~are~(in~alphabetic~order):~
10392     &~in~blocks,~
10393     allow-duplicate-names,~
10394     ampersand-in-blocks,~
10395     caption-above,~
10396     cell-space-bottom-limit,~
10397     cell-space-limits,~
10398     cell-space-top-limit,~
10399     code-for-first-col,~
10400     code-for-first-row,~
10401     code-for-last-col,~
10402     code-for-last-row,~
10403     corners,~
10404     custom-key,~
10405     create-extra-nodes,~
10406     create-medium-nodes,~
10407     create-large-nodes,~
10408     custom-line,~
10409     delimiters~(several~subkeys),~
10410     end-of-row,~
10411     first-col,~
10412     first-row,~
10413     hlines,~
10414     hvlines,~

```

```

10415     hvlines-except-borders,~
10416     last-col,~
10417     last-row,~
10418     left-margin,~
10419     light-syntax,~
10420     light-syntax-expanded,~
10421     matrix/columns-type,~
10422     no-cell-nodes,~
10423     notes~(several~subkeys),~
10424     nullify-dots,~
10425     pgf-node-code,~
10426     renew-dots,~
10427     renew-matrix,~
10428     respect-arraystretch,~
10429     rounded-corners,~
10430     right-margin,~
10431     rules~(with~the~subkeys~'color'~and~'width'),~
10432     small,~
10433     sub-matrix~(several~subkeys),~
10434     vl原因,~
10435     xdots~(several~subkeys).
10436 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10437 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10438 {
10439   Unknown~key.\\
10440   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10441   \{NiceArray\}. \\
10442   That~key~will~be~ignored. \\
10443   \c_@@_available_keys_str
10444 }
10445 {
10446   The~available~keys~are~(in~alphabetic~order):~
10447   &-in-blocks,~
10448   ampersand-in-blocks,~
10449   b,~
10450   baseline,~
10451   c,~
10452   cell-space-bottom-limit,~
10453   cell-space-limits,~
10454   cell-space-top-limit,~
10455   code-after,~
10456   code-for-first-col,~
10457   code-for-first-row,~
10458   code-for-last-col,~
10459   code-for-last-row,~
10460   columns-width,~
10461   corners,~
10462   create-extra-nodes,~
10463   create-medium-nodes,~
10464   create-large-nodes,~
10465   extra-left-margin,~
10466   extra-right-margin,~
10467   first-col,~
10468   first-row,~
10469   hlines,~
10470   hvlines,~
10471   hvlines-except-borders,~
10472   last-col,~
10473   last-row,~
10474   left-margin,~
10475   light-syntax,~

```

```

10476 light-syntax-expanded,~
10477 name,~
10478 no-cell-nodes,~
10479 nullify-dots,~
10480 pgf-node-code,~
10481 renew-dots,~
10482 respect-arraystretch,~
10483 right-margin,~
10484 rounded-corners,~
10485 rules~(with~the~subkeys~'color'~and~'width'),~
10486 small,~
10487 t,~
10488 vl原因,~
10489 xdots/color,~
10490 xdots/shorten-start,~
10491 xdots/shorten-end,~
10492 xdots/shorten-and~
10493 xdots/line-style.
10494 }

```

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`).

```

10495 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10496 {
10497   Unknown~key.\\
10498   The~key~'\l_keys_key_str'~is~unknown~for~the~
10499   \@@_full_name_env:. \\
10500   That~key~will~be~ignored. \\
10501   \c_@@_available_keys_str
10502 }
10503 {
10504   The~available~keys~are~(in~alphabetic~order):~
10505   &~in~blocks,~
10506   ampersand~in~blocks,~
10507   b,~
10508   baseline,~
10509   c,~
10510   cell-space-bottom-limit,~
10511   cell-space-limits,~
10512   cell-space-top-limit,~
10513   code~after,~
10514   code~for~first~col,~
10515   code~for~first~row,~
10516   code~for~last~col,~
10517   code~for~last~row,~
10518   columns~type,~
10519   columns~width,~
10520   corners,~
10521   create~extra~nodes,~
10522   create~medium~nodes,~
10523   create~large~nodes,~
10524   extra~left~margin,~
10525   extra~right~margin,~
10526   first~col,~
10527   first~row,~
10528   hlines,~
10529   hvlines,~
10530   hvlines~except~borders,~
10531   l,~
10532   last~col,~
10533   last~row,~
10534   left~margin,~
10535   light-syntax,~
10536   light-syntax-expanded,~

```



```

10537     name,~
10538     no-cell-nodes,~
10539     nullify-dots,~
10540     pgf-node-code,~
10541     r,~
10542     renew-dots,~
10543     respect-arraystretch,~
10544     right-margin,~
10545     rounded-corners,~
10546     rules~(with~the~subkeys~'color'~and~'width'),~
10547     small,~
10548     t,~
10549     vl原因,~
10550     xdots/color,~
10551     xdots/shorten-start,~
10552     xdots/shorten-end,~
10553     xdots/shorten-and~
10554     xdots/line-style.
10555 }
10556 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10557 {
10558     Unknown~key.\\
10559     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10560     \{NiceTabular\}. \\
10561     That~key~will~be~ignored. \\
10562     \c_@@_available_keys_str
10563 }
10564 {
10565     The~available~keys~are~(in~alphabetic~order):~
10566     &~in~blocks,~
10567     ampersand~in~blocks,~
10568     b,~
10569     baseline,~
10570     c,~
10571     caption,~
10572     cell-space-bottom-limit,~
10573     cell-space-limits,~
10574     cell-space-top-limit,~
10575     code~after,~
10576     code~for~first~col,~
10577     code~for~first~row,~
10578     code~for~last~col,~
10579     code~for~last~row,~
10580     columns~width,~
10581     corners,~
10582     custom~line,~
10583     create~extra~nodes,~
10584     create~medium~nodes,~
10585     create~large~nodes,~
10586     extra~left~margin,~
10587     extra~right~margin,~
10588     first~col,~
10589     first~row,~
10590     hlines,~
10591     hvlines,~
10592     hvlines~except~borders,~
10593     label,~
10594     last~col,~
10595     last~row,~
10596     left~margin,~
10597     light~syntax,~
10598     light~syntax~expanded,~
10599     name,~

```

```

10600 no-cell-nodes,~
10601 notes~(several~subkeys),~
10602 nullify-dots,~
10603 pgf-node-code,~
10604 renew-dots,~
10605 respect-arraystretch,~
10606 right-margin,~
10607 rounded-corners,~
10608 rules~(with~the~subkeys~'color'~and~'width'),~
10609 short-caption,~
10610 t,~
10611 tabularnote,~
10612 vlines,~
10613 xdots/color,~
10614 xdots/shorten-start,~
10615 xdots/shorten-end,~
10616 xdots/shorten-and~
10617 xdots/line-style.
10618 }

10619 \@@_msg_new:nnn { Duplicate~name }
10620 {
10621   Duplicate~name.\\
10622   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10623   the~same~environment~name~twice.~You~can~go~on,~but,~
10624   maybe,~you~will~have~incorrect~results~especially~
10625   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10626   message~again,~use~the~key~'allow-duplicate-names'~in~
10627   '\token_to_str:N \NiceMatrixOptions'.\\
10628   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10629     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10630 }
10631 {
10632   The~names~already~defined~in~this~document~are:~
10633   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10634 }

10635 \@@_msg_new:nn { Option~auto~for~columns-width }
10636 {
10637   Erroneous~use.\\
10638   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10639   That~key~will~be~ignored.
10640 }

10641 \@@_msg_new:nn { NiceTabularX~without~X }
10642 {
10643   NiceTabularX~without~X.\\
10644   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10645   However,~you~can~go~on.
10646 }

10647 \@@_msg_new:nn { Preamble~forgotten }
10648 {
10649   Preamble~forgotten.\\
10650   You~have~probably~forgotten~the~preamble~of~your~
10651   \@@_full_name_env:. \\
10652   This~error~is~fatal.
10653 }

10654 \@@_msg_new:nn { Invalid~col~number }
10655 {
10656   Invalid~column~number.\\
10657   A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10658   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10659 }

10660 \@@_msg_new:nn { Invalid~row~number }

```

```
10661 {
10662     Invalid~row~number.\\
10663     A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10664     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10665 }
```

# Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	19
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	74
13	The environment <code>{NiceMatrix}</code> and its variants	92
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
15	After the construction of the array	94
16	We draw the dotted lines	101
17	The actual instructions for drawing the dotted lines with Tikz	114
18	User commands available in the new environments	120
19	The command <code>\line</code> accessible in code-after	126
20	The command <code>\RowStyle</code>	128
21	Colors of cells, rows and columns	131
22	The vertical and horizontal rules	143
23	The empty corners	159
24	The environment <code>{NiceMatrixBlock}</code>	162
25	The extra nodes	163
26	The blocks	168
27	How to draw the dotted lines transparently	192
28	Automatic arrays	192
29	The redefinition of the command <code>\dotfill</code>	193
30	The command <code>\diagbox</code>	194

31	The keyword <code>\CodeAfter</code>	195
32	The delimiters in the preamble	196
33	The command <code>\SubMatrix</code>	197
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	206
35	The commands <code>HBrace</code> et <code>VBrace</code>	209
36	The command <code>TikzEveryCell</code>	212
37	The command <code>\ShowCellNames</code>	213
38	We process the options at package loading	215
39	About the package underscore	217
40	Error messages of the package	217