

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 12, 2024

Abstract

This document is the documented code of the LaTeX package `nicematrix`. It is *not* its user's guide. The guide of utilisation is the document `nicematrix.pdf` (with a French traduction: `nicematrix-french.pdf`).

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

1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5   {nicematrix}
6   {\myfiledate}
7   {\myfileversion}
8   {Enhanced arrays with the help of PGF/TikZ}
9 \ProvideDocumentCommand{\IfPackageLoadedT}{mm}
10  {\IfPackageLoadedTF{#1}{#2}{}}
11
12 \ProvideDocumentCommand{\IfPackageLoadedF}{mm}
13  {\IfPackageLoadedTF{#1}{}{#2}}
```

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 6.29b of `nicematrix`, at the date of 2024/11/12.

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

```

16 \bool_const:Nn \c_@@_tagging_array_bool
17 { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
18 \bool_const:Nn \c_@@_testphase_table_bool
19 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool
20 }

21 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
22 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
23 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
24 \cs_generate_variant:Nn \@@_error:nn { n e }
25 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
26 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
27 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
28 \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).

```

29 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
30 {
31   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
32     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
33     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
34 }

```

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

```

35 \cs_new_protected:Npn \@@_error_or_warning:n
36 { \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".

```

37 \bool_new:N \g_@@_messages_for_Overleaf_bool
38 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
39 {
40   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
41   || \str_if_eq_p:ee \c_sys_jobname_str { output } % for Overleaf
42 }

```

```

43 \cs_new_protected:Npn \@@_msg_redirect_name:nn
44 { \msg_redirect_name:nnn { nicematrix } }
45 \cs_new_protected:Npn \@@_gredirect_none:n #1
46 {
47   \group_begin:
48   \globaldefs = 1
49   \@@_msg_redirect_name:nn { #1 } { none }
50   \group_end:
51 }
52 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
53 {
54   \@@_error:n { #1 }
55   \@@_gredirect_none:n { #1 }
56 }
57 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
58 {
59   \@@_warning:n { #1 }
60   \@@_gredirect_none:n { #1 }
61 }

```

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

```
62 \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
63 \cs_set:Npn \int_if_zero:NTF #1 { \int_compare:nNnTF #1 = \c_zero_int }

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

69 \hook_gput_code:nnn { begindocument / end } { . }
70 { \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.

Exemple :

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

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

```
71 \cs_new_protected:Npn \@@_collect_options:n #1
72 {
73   \peek_meaning:NTF [
74     { \@@_collect_options:nw { #1 } }
75     { #1 { } }
76 }
```

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

```
77 \NewDocumentCommand \@@_collect_options:nw { m r[] }
78 { \@@_collect_options:nn { #1 } { #2 } }
79
80 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
81 {
82   \peek_meaning:NTF [
83     { \@@_collect_options:nw { #1 } { #2 } }
84     { #1 { #2 } }
85 }
86
87 \cs_new_protected:Npn \@@_collect_options:nw #1#2[#3]
88 { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

3 Technical definitions

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

```

89 \tl_const:Nn \c_@@_b_tl { b }
90 \tl_const:Nn \c_@@_c_tl { c }
91 \tl_const:Nn \c_@@_l_tl { l }
92 \tl_const:Nn \c_@@_r_tl { r }
93 \tl_const:Nn \c_@@_all_tl { all }
94 \tl_const:Nn \c_@@_dot_tl { . }
95 \str_const:Nn \c_@@_r_str { r }
96 \str_const:Nn \c_@@_c_str { c }
97 \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).

```

98 \tl_new:N \l_@@_argspec_tl

99 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
100 \cs_generate_variant:Nn \str_lowercase:n { o }
101 \cs_generate_variant:Nn \str_set:Nn { N o }
102 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
103 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
104 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
105 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
106 \cs_generate_variant:Nn \dim_min:nn { v }
107 \cs_generate_variant:Nn \dim_max:nn { v }

108 \hook_gput_code:nnn { begindocument } { . }
109 {
110   \IfPackageLoadedTF { tikz }
111   {

```

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.

```

112   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
113   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
114   }
115   {
116   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
117   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
118   }
119 }

```

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`).

```

120 \IfClassLoadedTF { revtex4-1 }
121 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
122 {
123   \IfClassLoadedTF { revtex4-2 }
124   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
125   {

```

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

```

126     \cs_if_exist:NT \rvtx@ifformat@geq
127     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
128     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
129   }
130 }

```

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.

```

131 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
132 {
133   \iow_now:Nn \@mainaux
134   {
135     \ExplSyntaxOn
136     \cs_if_free:NT \pgfsyspdfmark
137     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
138     \ExplSyntaxOff
139   }
140   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
141 }

```

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

```

142 \ProvideDocumentCommand \iddots { }
143 {
144   \mathinner
145   {
146     \tex_mkern:D 1 mu
147     \box_move_up:nn { 1 pt } { \hbox { . } }
148     \tex_mkern:D 2 mu
149     \box_move_up:nn { 4 pt } { \hbox { . } }
150     \tex_mkern:D 2 mu
151     \box_move_up:nn { 7 pt }
152     { \vbox:n { \kern 7 pt \hbox { . } } }
153     \tex_mkern:D 1 mu
154   }
155 }

```

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.

```

156 \hook_gput_code:nnn { begindocument } { . }
157 {
158   \IfPackageLoadedT { booktabs }
159   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
160 }
161 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
162 {
163   \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`).

```

164   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
165   {

```

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

```
166     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
167     { \@@_old_pgfulfil@check@rerun { ##1 } { ##2 } }
168   }
169 }
```

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

```
170 \hook_gput_code:nnn { begindocument } { . }
171 {
172   \IfPackageLoadedF { colortbl }
173   {
```

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.

```
174     \cs_set_protected:Npn \CT@arc@ { }
175     \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
176     \cs_set_nopar:Npn \CT@arc #1 #2
177     {
178       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
179       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
180     }
```

Idem for `\CT@drs@`.

```
181     \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
182     \cs_set_nopar:Npn \CT@drs #1 #2
183     {
184       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
185       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
186     }
187     \cs_set_nopar:Npn \hline
188     {
189       \noalign { \ifnum 0 = ` } \fi
190       \cs_set_eq:NN \hskip \vskip
191       \cs_set_eq:NN \vrule \hrule
192       \cs_set_eq:NN \@width \@height
193       { \CT@arc@ \vline }
194       \futurelet \reserved@a
195       \@xhline
196     }
197   }
198 }
```

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.

```
199 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
200 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
201 {
202   \int_if_zero:nT \l_@@_first_col_int { \omit & }
203   \int_compare:nNnT { #1 } > \c_one_int
204   { \multispan { \int_eval:n { #1 - 1 } } & }
205   \multispan { \int_eval:n { #2 - #1 + 1 } }
206   {
207     \CT@arc@
208     \leaders \hrule \@height \arrayrulewidth \hfill
```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`¹

```
209     \skip_horizontal:N \c_zero_dim
210   }
```

¹See 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`.

```

211   \everycr { }
212   \cr
213   \noalign { \skip_vertical:N -\arrayrulewidth }
214 }

```

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.

```

215 \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`.

```

216 { \@@_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*.

```

217 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
218 \cs_generate_variant:Nn \@@_cline_i:nn { e }
219 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
220 {
221   \tl_if_empty:nTF { #3 }
222     { \@@_cline_iii:w #1|#2-#2 \q_stop }
223     { \@@_cline_ii:w #1|#2-#3 \q_stop }
224 }
225 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
226 { \@@_cline_iii:w #1|#2-#3 \q_stop }
227 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
228 {

```

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).

```

229   \int_compare:nNnT { #1 } < { #2 }
230     { \multispan { \int_eval:n { #2 - #1 } } & }
231   \multispan { \int_eval:n { #3 - #2 + 1 } }
232   {
233     \CT@arc@
234     \leaders \hrule \@height \arrayrulewidth \hfill
235     \skip_horizontal:N \c_zero_dim
236   }

```

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

```

237   \peek_meaning_remove_ignore_spaces:NTF \cline
238     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
239     { \everycr { } \cr }
240 }

```

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

```

241 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

242 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
243 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
244 {
245   \tl_if_blank:nF { #1 }
246   {
247     \tl_if_head_eq_meaning:nNTF { #1 } [
248       { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
249       { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
250     ]
251 }

```

```

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

```

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

```

259 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
260 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
261 {
262   \tl_if_head_eq_meaning:nNTF { #2 } [
263     { #1 #2 }
264     { #1 { #2 } }
265   ]

```

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

```

266 \cs_generate_variant:Nn \@@_color:n { o }
267 \cs_new_protected:Npn \@@_color:n #1
268 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

```

```

269 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
270 {
271   \tl_set_rescan:Nno
272     #1
273     {
274       \char_set_catcode_other:N >
275       \char_set_catcode_other:N <
276     }
277   #1
278 }

```

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.

```

279 \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).

```

280 \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.

```

281 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
282 { \int_use:N \g_@@_env_int }

```

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

```

283 \cs_new_protected:Npn \@@_qpoint:n #1
284 { \pgfpointanchor { \@@_env: - #1 } { center } }

```


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

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

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

```
286 \bool_new:N \g_@@_delims_bool
287 \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.

```
288 \bool_new:N \l_@@_preamble_bool
289 \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.

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

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

```
291 \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.

```
292 \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).

```
293 \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.).

```
294 \dim_new:N \l_@@_col_width_dim
295 \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.

```
296 \int_new:N \g_@@_row_total_int
297 \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).

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

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

```
299 \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.

```
300 \tl_new:N \l_@@_hpos_cell_tl
301 \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`.

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

Idem for the mono-row blocks.

```
303 \dim_new:N \g_@@_blocks_ht_dim
304 \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}`).

```
305 \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`.

```
306 \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.

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

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

```
308 \bool_new:N \l_@@_notes_detect_duplicates_bool
309 \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.

```
310 \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).

```
311 \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).

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

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

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

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

```
314 \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.

```
315 \bool_new:N \l_@@_X_bool
316 \bool_new:N \g_@@_caption_finished_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The contain 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 }`).

```
317 \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.

```
318 \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.

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

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

```
321 \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}`).

```
322 \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`).

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

For `\multicolumn`.

```
324 \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`.

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

```
326 \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 `:`.

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

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

```
329 \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`).

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

```
331 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
332 {
```

```
333   \if_mode_math: \else:
```

```
334     \@@_fatal:n { Outside-math-mode }
```

```
335   \fi:
```

```
336 }
```

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.

```
337 \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”.

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

```
339 \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`).

```
340 \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*.

```

341 \tl_new:N \g_@@_com_or_env_str
342 \tl_gset:Nn \g_@@_com_or_env_str { environment }

343 \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)`.

```

344 \cs_new:Npn \@@_full_name_env:
345 {
346   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
347   { command \space \c_backslash_str \g_@@_name_env_str }
348   { environment \space \{ \g_@@_name_env_str \} }
349 }

```

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

```

350 \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.

```

351 \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.

```

352 \tl_new:N \g_@@_pre_code_before_tl
353 \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.

```

354 \tl_new:N \g_@@_pre_code_after_tl
355 \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.

```

356 \bool_new:N \l_@@_in_code_after_bool

```

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

```

357 \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.

```

358 \int_new:N \l_@@_old_iRow_int
359 \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).

```

360 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```
361 \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.

```
362 \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.

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

```
364 \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.

```
365 \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).

```
366 \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).

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

The use of `\l_@@_code_before_tl` is not clear. Maybe that with the evolutions of `nicematrix`, it has become obsolete. We should have a look at that.

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

```
369 \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).

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

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

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

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

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

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

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

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

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

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

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

```
379 \dim_new:N \g_@@_dp_row_zero_dim
```

```
380 \dim_new:N \g_@@_ht_row_zero_dim
```

```
381 \dim_new:N \g_@@_ht_row_one_dim
```

```
382 \dim_new:N \g_@@_dp_ante_last_row_dim
```

```
383 \dim_new:N \g_@@_ht_last_row_dim
```

```
384 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
385 \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”.

```
386 \dim_new:N \g_@@_width_last_col_dim
387 \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.

```
388 \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!).

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

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

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

```
390 \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.

```
391 \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`).

```
392 \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`).

```
393 \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.

```
394 \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) correspondent will be stored. These lists will be used for the creation of the “medium nodes” (if they are created).

```
395 \seq_new:N \g_@@_multicolumn_cells_seq
396 \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).

```
397 \int_new:N \l_@@_row_min_int
398 \int_new:N \l_@@_row_max_int
399 \int_new:N \l_@@_col_min_int
400 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
401 \int_new:N \l_@@_start_int
402 \int_set_eq:NN \l_@@_start_int \c_one_int
403 \int_new:N \l_@@_end_int
404 \int_new:N \l_@@_local_start_int
405 \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.

```
406 \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).

```
407 \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`.

```
408 \tl_new:N \l_@@_fill_tl
409 \tl_new:N \l_@@_opacity_tl
410 \tl_new:N \l_@@_draw_tl
411 \seq_new:N \l_@@_tikz_seq
412 \clist_new:N \l_@@_borders_clist
413 \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.

```
414 \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`.

```
415 \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.

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

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

```
417 \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`).

```
418 \str_new:N \l_@@_hpos_block_str
419 \str_set:Nn \l_@@_hpos_block_str { c }
420 \bool_new:N \l_@@_hpos_of_block_cap_bool
421 \bool_new:N \l_@@_p_block_bool
```

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

```
422 \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).

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

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

```
424 \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).

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

```
426 \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.

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

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

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

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

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

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

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

```
434 \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}`).

```
435 \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.

```
436 \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`).

```
437 \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.

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

```
439 \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.

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

```
441 \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).

```
442 \int_new:N \l_@@_last_row_int
443 \int_set:Nn \l_@@_last_row_int { -2 }
```

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
445 \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`.

```
446 \int_new:N \l_@@_last_col_int
447 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
448 \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`).

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

Some utilities

```
450 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
451 {
```

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

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

```

452   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
453   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
454 }

```

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.

```

455 \cs_new_protected:Npn \@_expand_clist:N #1
456 {
457   \clist_if_in:NnF #1 { all }
458   {
459     \clist_clear:N \l_tmpa_clist
460     \clist_map_inline:Nn #1
461     {

```

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

```

462       \tl_if_in:nnTF { ##1 } { - }
463       { \@_cut_on_hyphen:w ##1 \q_stop }
464       {

```

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

```

465           \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
466           \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
467       }
468       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
469       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
470   }
471   \tl_set_eq:NN #1 \l_tmpa_clist
472 }
473 }

```

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.

```

474 \hook_gput_code:nnn { begindocument } { . }
475 {
476   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
477   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
478   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
479 }

```

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`.³
 - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\c_novalue_tl`).
 - 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.

```
480 \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`.

```
481 \int_new:N \g_@@_tabularnote_int
482 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
483 \seq_new:N \g_@@_notes_seq
484 \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.

```
485 \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.

```
486 \seq_new:N \l_@@_notes_labels_seq
487 \newcounter { nicematrix_draft }
488 \cs_new_protected:Npn \@@_notes_format:n #1
489 {
490   \setcounter { nicematrix_draft } { #1 }
491   \@@_notes_style:n { nicematrix_draft }
492 }
```

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

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

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

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

```
494 \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`.

```
495 \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`.

```
496 \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`).

```
497 \hook_gput_code:nnn { begindocument } { . }
498 {
499   \IfPackageLoadedTF { enumitem }
500   {
```

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.

```
501     \newlist { tabularnotes } { enumerate } { 1 }
502     \setlist [ tabularnotes ]
503     {
504       topsep = 0pt ,
505       noitemsep ,
506       leftmargin = * ,
507       align = left ,
508       labelsep = 0pt ,
509       label =
510         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
511     }
512     \newlist { tabularnotes* } { enumerate* } { 1 }
513     \setlist [ tabularnotes* ]
514     {
515       afterlabel = \nobreak ,
516       itemjoin = \quad ,
517       label =
518         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
519     }
```

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`.

```
520     \NewDocumentCommand \tabularnote { o m }
521     {
522       \bool_lazy_or:nnT { \cs_if_exist_p:N \@capytype } \l_@@_in_env_bool
523       {
524         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
525         { \@@_error:n { tabularnote-forbidden } }
526         {
527           \bool_if:NTF \l_@@_in_caption_bool
528             \@@_tabularnote_caption:nn
529             \@@_tabularnote:nn
530             { #1 } { #2 }
531         }
532     }
```

```

533     }
534   }
535   {
536     \NewDocumentCommand \tabularnote { o m }
537     {
538       \@@_error_or_warning:n { enumitem-not-loaded }
539       \@@_gredirect_none:n { enumitem-not-loaded }
540     }
541   }
542 }

543 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
544 { \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`.

```

545 \cs_new_protected:Npn \@@_tabularnote:mn #1 #2
546 {

```

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).

```

547   \int_zero:N \l_tmpa_int
548   \bool_if:NT \l_@@_notes_detect_duplicates_bool
549   {

```

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`.

```

550     \int_zero:N \l_tmpb_int
551     \seq_map_indexed_inline:Nn \g_@@_notes_seq
552     {
553       \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
554       \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
555       {
556         \tl_if_novalue:nTF { #1 }
557         { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
558         { \int_set:Nn \l_tmpa_int { ##1 } }
559         \seq_map_break:
560       }
561     }
562     \int_if_zero:nF \l_tmpa_int
563     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
564   }
565   \int_if_zero:nT \l_tmpa_int
566   {
567     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
568     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
569   }
570   \seq_put_right:Ne \l_@@_notes_labels_seq
571   {
572     \tl_if_novalue:nTF { #1 }
573     {
574       \@@_notes_format:n
575       {
576         \int_eval:n

```

```

577         {
578         \int_if_zero:nTF \l_tmpa_int
579         \c@tabularnote
580         \l_tmpa_int
581         }
582     }
583 }
584 { #1 }
585 }
586 \peek_meaning:Nf \tabularnote
587 {

```

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`.

```

588     \hbox_set:Nn \l_tmpa_box
589     {

```

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

```

590     \@@_notes_label_in_tabular:n
591     {
592     \seq_use:Nnnn
593     \l_@@_notes_labels_seq { , } { , } { , }
594     }
595 }

```

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

```

596     \int_gdecr:N \c@tabularnote
597     \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.

```

598     \int_gincr:N \g_@@_tabularnote_int
599     \refstepcounter { tabularnote }
600     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
601     { \int_gincr:N \c@tabularnote }
602     \seq_clear:N \l_@@_notes_labels_seq
603     \bool_lazy_or:nnTF
604     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
605     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
606     {
607     \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).

```

608     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
609     }
610     { \box_use:N \l_tmpa_box }
611 }
612 }

```

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`.

```

613 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
614 {
615     \bool_if:Nf \g_@@_caption_finished_bool
616     {

```

```

617     \int_compare:nNnT \c@tabulernote = \g_@@_notes_caption_int
618     { \int_gzero:N \c@tabulernote }

```

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`!

```

619     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
620     { \@@_error:n { Identical-notes-in-caption } }
621   }
622   {

```

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

```

623     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
624     {

```

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 `\tabulernote` in the caption.

```

625         \bool_gset_true:N \g_@@_caption_finished_bool
626         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
627         \int_gzero:N \c@tabulernote
628     }
629     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
630   }

```

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!

```

631     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
632     \seq_put_right:Ne \l_@@_notes_labels_seq
633     {
634         \tl_if_novalue:nTF { #1 }
635         { \@@_notes_format:n { \int_use:N \c@tabulernote } }
636         { #1 }
637     }
638     \peek_meaning:NF \tabulernote
639     {
640         \@@_notes_label_in_tabular:n
641         { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
642         \seq_clear:N \l_@@_notes_labels_seq
643     }
644   }

645 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
646 { \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.

```

647 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
648 {
649     \begin { pgfscope }
650     \pgfset
651     {
652         inner~sep = \c_zero_dim ,
653         minimum~size = \c_zero_dim
654     }
655     \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
656     \pgfnode
657     { rectangle }

```

```

658     { center }
659     {
660     \vbox_to_ht:nn
661     { \dim_abs:n { #5 - #3 } }
662     {
663     \vfill
664     \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
665     }
666     }
667     { #1 }
668     { }
669 \end { pgfscope }
670 }

```

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.

```

671 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
672 {
673 \begin { pgfscope }
674 \pgfset
675 {
676 inner~sep = \c_zero_dim ,
677 minimum~size = \c_zero_dim
678 }
679 \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
680 \pgfpointdiff { #3 } { #2 }
681 \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
682 \pgfnode
683 { rectangle }
684 { center }
685 {
686 \vbox_to_ht:nn
687 { \dim_abs:n \l_tmpb_dim }
688 { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
689 }
690 { #1 }
691 { }
692 \end { pgfscope }
693 }

```

7 The options

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

```

694 \tl_new:N \l_@@_caption_tl
695 \tl_new:N \l_@@_short_caption_tl
696 \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).

```

697 \bool_new:N \l_@@_caption_above_bool

```

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

```

698 \bool_new:N \l_@@_color_inside_bool

```


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

```
699 \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`).

```
700 \dim_new:N \l_@@_cell_space_top_limit_dim
701 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
702 \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.

```
703 \dim_new:N \l_@@_xdots_inter_dim
704 \hook_gput_code:nnn { begindocument } { . }
705 { \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).

```
706 \dim_new:N \l_@@_xdots_shorten_start_dim
707 \dim_new:N \l_@@_xdots_shorten_end_dim
708 \hook_gput_code:nnn { begindocument } { . }
709 {
710   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
711   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
712 }
```

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.

```
713 \dim_new:N \l_@@_xdots_radius_dim
714 \hook_gput_code:nnn { begindocument } { . }
715 { \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.

```
716 \tl_new:N \l_@@_xdots_line_style_tl
717 \tl_const:Nn \c_@@_standard_tl { standard }
718 \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`.

```
719 \bool_new:N \l_@@_light_syntax_bool
720 \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).

```
721 \tl_new:N \l_@@_baseline_tl
722 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
723 \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`).

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

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

```
725 \bool_new:N \l_@@_parallelize_diags_bool
726 \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.

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

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

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

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

```
732 \cs_new_protected:Npn \@@_reset_arraystretch:
733 { \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).

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

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

```
735 \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.

```
736 \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”.

```
737 \bool_new:N \l_@@_medium_nodes_bool
738 \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).

```
739 \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`).

```
740 \dim_new:N \l_@@_left_margin_dim
741 \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`.

```
742 \dim_new:N \l_@@_extra_left_margin_dim
743 \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.

```
744 \tl_new:N \l_@@_end_of_row_tl
745 \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 “:”.

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

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

```
747 \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 acheive 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.

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

```
749 \keys_define:nn { nicematrix / xdots }
750 {
751   shorten-start .code:n =
752     \hook_gput_code:nnn { begindocument } { . }
753     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
754   shorten-end .code:n =
755     \hook_gput_code:nnn { begindocument } { . }
756     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
757   shorten-start .value_required:n = true ,
758   shorten-end .value_required:n = true ,
759   shorten .code:n =
760     \hook_gput_code:nnn { begindocument } { . }
761     {
762       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
763       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
764     } ,
765   shorten .value_required:n = true ,
766   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
767   horizontal-labels .default:n = true ,
768   line-style .code:n =
769     {
770       \bool_lazy_or:nnTF
771         { \cs_if_exist_p:N \tikzpicture }
772         { \str_if_eq_p:mn { #1 } { standard } }
773         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
774         { \@@_error:n { bad-option-for-line-style } }
775     } ,
```

```

776 line-style .value_required:n = true ,
777 color .tl_set:N = \l_@@_xdots_color_tl ,
778 color .value_required:n = true ,
779 radius .code:n =
780   \hook_gput_code:nnn { begindocument } { . }
781   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
782 radius .value_required:n = true ,
783 inter .code:n =
784   \hook_gput_code:nnn { begindocument } { . }
785   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
786 radius .value_required:n = true ,

```

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

```

787 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
788 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
789 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).

```

790 draw-first .code:n = \prg_do_nothing: ,
791 unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
792 }

```

```

793 \keys_define:nn { nicematrix / rules }
794 {
795   color .tl_set:N = \l_@@_rules_color_tl ,
796   color .value_required:n = true ,
797   width .dim_set:N = \arrayrulewidth ,
798   width .value_required:n = true ,
799   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
800 }

```

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

```

801 \keys_define:nn { nicematrix / Global }
802 {
803   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
804   ampersand-in-blocks .default:n = true ,
805   &-in-blocks .meta:n = ampersand-in-blocks ,
806   no-cell-nodes .code:n =
807     \cs_set_protected:Npn \@@_node_for_cell:
808     { \box_use_drop:N \l_@@_cell_box } ,
809   no-cell-nodes .value_forbidden:n = true ,
810   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
811   rounded-corners .default:n = 4 pt ,
812   custom-line .code:n = \@@_custom_line:n { #1 } ,
813   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
814   rules .value_required:n = true ,
815   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
816   standard-cline .default:n = true ,
817   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
818   cell-space-top-limit .value_required:n = true ,
819   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
820   cell-space-bottom-limit .value_required:n = true ,
821   cell-space-limits .meta:n =
822     {
823       cell-space-top-limit = #1 ,
824       cell-space-bottom-limit = #1 ,
825     } ,

```

```

826 cell-space-limits .value_required:n = true ,
827 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
828 light-syntax .code:n =
829   \bool_set_true:N \l_@@_light_syntax_bool
830   \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
831 light-syntax .value_forbidden:n = true ,
832 light-syntax-expanded .code:n =
833   \bool_set_true:N \l_@@_light_syntax_bool
834   \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
835 light-syntax-expanded .value_forbidden:n = true ,
836 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
837 end-of-row .value_required:n = true ,
838 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
839 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
840 last-row .int_set:N = \l_@@_last_row_int ,
841 last-row .default:n = -1 ,
842 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
843 code-for-first-col .value_required:n = true ,
844 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
845 code-for-last-col .value_required:n = true ,
846 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
847 code-for-first-row .value_required:n = true ,
848 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
849 code-for-last-row .value_required:n = true ,
850 hlines .clist_set:N = \l_@@_hlines_clist ,
851 vlines .clist_set:N = \l_@@_vlines_clist ,
852 hlines .default:n = all ,
853 vlines .default:n = all ,
854 vlines-in-sub-matrix .code:n =
855   {
856     \tl_if_single_token:nTF { #1 }
857     {
858       \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
859       { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

860     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
861   }
862   { \@@_error:n { One-letter-allowed } }
863 } ,
864 vlines-in-sub-matrix .value_required:n = true ,
865 hvlines .code:n =
866   {
867     \bool_set_true:N \l_@@_hvlines_bool
868     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
869     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
870   } ,
871 hvlines-except-borders .code:n =
872   {
873     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
874     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
875     \bool_set_true:N \l_@@_hvlines_bool
876     \bool_set_true:N \l_@@_except_borders_bool
877   } ,
878 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.

```

879 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
880 renew-dots .value_forbidden:n = true ,
881 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
882 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
883 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,

```

```

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

```

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

```

903 \keys_define:nn { nicematrix / environments }
904 {
905   corners .clist_set:N = \l_@@_corners_clist ,
906   corners .default:n = { NW , SW , NE , SE } ,
907   code-before .code:n =
908     {
909       \tl_if_empty:nF { #1 }
910       {
911         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
912         \bool_set_true:N \l_@@_code_before_bool
913       }
914     } ,
915   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}`.

```

916   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
917   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
918   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
919   baseline .tl_set:N = \l_@@_baseline_tl ,
920   baseline .value_required:n = true ,
921   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)`.

```

922   \str_if_eq:eeTF { #1 } { auto }
923   { \bool_set_true:N \l_@@_auto_columns_width_bool }
924   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
925   columns-width .value_required:n = true ,
926   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.

```

927   \legacy_if:nF { measuring@ }
928   {
929     \str_set:Ne \l_tmpa_str { #1 }
930     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
931     { \@@_error:nn { Duplicate~name } { #1 } }
932     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
933     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
934   } ,

```

```

935     name .value_required:n = true ,
936     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
937     code-after .value_required:n = true ,
938     color-inside .code:n =
939         \bool_set_true:N \l_@@_color_inside_bool
940         \bool_set_true:N \l_@@_code_before_bool ,
941     color-inside .value_forbidden:n = true ,
942     colortbl-like .meta:n = color-inside
943 }
944 \keys_define:nn { nicematrix / notes }
945 {
946     para .bool_set:N = \l_@@_notes_para_bool ,
947     para .default:n = true ,
948     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
949     code-before .value_required:n = true ,
950     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
951     code-after .value_required:n = true ,
952     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
953     bottomrule .default:n = true ,
954     style .cs_set:Np = \@@_notes_style:n #1 ,
955     style .value_required:n = true ,
956     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
957     label-in-tabular .value_required:n = true ,
958     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
959     label-in-list .value_required:n = true ,
960     enumitem-keys .code:n =
961     {
962         \hook_gput_code:nnn { begindocument } { . }
963         {
964             \IfPackageLoadedT { enumitem }
965             { \setlist* [ tabularnotes ] { #1 } }
966         }
967     } ,
968     enumitem-keys .value_required:n = true ,
969     enumitem-keys-para .code:n =
970     {
971         \hook_gput_code:nnn { begindocument } { . }
972         {
973             \IfPackageLoadedT { enumitem }
974             { \setlist* [ tabularnotes* ] { #1 } }
975         }
976     } ,
977     enumitem-keys-para .value_required:n = true ,
978     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
979     detect-duplicates .default:n = true ,
980     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
981 }
982 \keys_define:nn { nicematrix / delimiters }
983 {
984     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
985     max-width .default:n = true ,
986     color .tl_set:N = \l_@@_delimiters_color_tl ,
987     color .value_required:n = true ,
988 }

```

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

```

989 \keys_define:nn { nicematrix }
990 {
991     NiceMatrixOptions .inherit:n =
992     { nicematrix / Global } ,
993     NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

994 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
995 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
996 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
997 SubMatrix / rules .inherit:n = nicematrix / rules ,
998 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
999 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1000 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1001 NiceMatrix .inherit:n =
1002   {
1003     nicematrix / Global ,
1004     nicematrix / environments ,
1005   } ,
1006 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1007 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1008 NiceTabular .inherit:n =
1009   {
1010     nicematrix / Global ,
1011     nicematrix / environments
1012   } ,
1013 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1014 NiceTabular / rules .inherit:n = nicematrix / rules ,
1015 NiceTabular / notes .inherit:n = nicematrix / notes ,
1016 NiceArray .inherit:n =
1017   {
1018     nicematrix / Global ,
1019     nicematrix / environments ,
1020   } ,
1021 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1022 NiceArray / rules .inherit:n = nicematrix / rules ,
1023 pNiceArray .inherit:n =
1024   {
1025     nicematrix / Global ,
1026     nicematrix / environments ,
1027   } ,
1028 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1029 pNiceArray / rules .inherit:n = nicematrix / rules ,
1030 }

```

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

```

1031 \keys_define:nn { nicematrix / NiceMatrixOptions }
1032   {
1033     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1034     delimiters / color .value_required:n = true ,
1035     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1036     delimiters / max-width .default:n = true ,
1037     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1038     delimiters .value_required:n = true ,
1039     width .dim_set:N = \l_@@_width_dim ,
1040     width .value_required:n = true ,
1041     last-col .code:n =
1042       \tl_if_empty:nF { #1 }
1043         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1044       \int_zero:N \l_@@_last_col_int ,
1045     small .bool_set:N = \l_@@_small_bool ,
1046     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.

```

1047     renew-matrix .code:n = \@@_renew_matrix: ,
1048     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}`.

```
1049 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.

```
1050 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)`.

```
1051 \str_if_eq:eeTF { #1 } { auto }
1052 { \@@_error:n { Option~auto~for~columns~width } }
1053 { \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.

```
1054 allow-duplicate-names .code:n =
1055   \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1056 allow-duplicate-names .value_forbidden:n = true ,
1057 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1058 notes .value_required:n = true ,
1059 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1060 sub-matrix .value_required:n = true ,
1061 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1062 matrix / columns-type .value_required:n = true ,
1063 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1064 caption-above .default:n = true ,
1065 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1066 }
```

`\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.

```
1067 \NewDocumentCommand \NiceMatrixOptions { m }
1068 { \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.

```
1069 \keys_define:nn { nicematrix / NiceMatrix }
1070 {
1071   last-col .code:n = \tl_if_empty:nTF { #1 }
1072     {
1073       \bool_set_true:N \l_@@_last_col_without_value_bool
1074       \int_set:Nn \l_@@_last_col_int { -1 }
1075     }
1076     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1077   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1078   columns-type .value_required:n = true ,
1079   l .meta:n = { columns-type = l } ,
1080   r .meta:n = { columns-type = r } ,
1081   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1082   delimiters / color .value_required:n = true ,
1083   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1084   delimiters / max-width .default:n = true ,
1085   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1086   delimiters .value_required:n = true ,
1087   small .bool_set:N = \l_@@_small_bool ,
1088   small .value_forbidden:n = true ,
1089   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1090 }
```

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

```
1091 \keys_define:nn { nicematrix / NiceArray }
1092 {
```

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.

```
1093     small .bool_set:N = \l_@@_small_bool ,
1094     small .value_forbidden:n = true ,
1095     last-col .code:n = \tl_if_empty:nF { #1 }
1096                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1097                 \int_zero:N \l_@@_last_col_int ,
1098     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1099     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1100     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1101 }
```

```
1102 \keys_define:nn { nicematrix / pNiceArray }
1103 {
1104     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1105     last-col .code:n = \tl_if_empty:nF { #1 }
1106                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1107                 \int_zero:N \l_@@_last_col_int ,
1108     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1109     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1110     delimiters / color .value_required:n = true ,
1111     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1112     delimiters / max-width .default:n = true ,
1113     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1114     delimiters .value_required:n = true ,
1115     small .bool_set:N = \l_@@_small_bool ,
1116     small .value_forbidden:n = true ,
1117     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1118     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1119     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1120 }
```

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

```
1121 \keys_define:nn { nicematrix / NiceTabular }
1122 {
```

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.

```
1123     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1124                 \bool_set_true:N \l_@@_width_used_bool ,
1125     width .value_required:n = true ,
1126     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1127     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1128     tabularnote .value_required:n = true ,
1129     caption .tl_set:N = \l_@@_caption_tl ,
1130     caption .value_required:n = true ,
1131     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1132     short-caption .value_required:n = true ,
1133     label .tl_set:N = \l_@@_label_tl ,
1134     label .value_required:n = true ,
1135     last-col .code:n = \tl_if_empty:nF { #1 }
1136                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1137                 \int_zero:N \l_@@_last_col_int ,
1138     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1139     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1140     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1141 }
```

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

1142 \keys_define:nn { nicematrix / CodeAfter }
1143 {
1144   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1145   delimiters / color .value_required:n = true ,
1146   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1147   rules .value_required:n = true ,
1148   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1149   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1150   sub-matrix .value_required:n = true ,
1151   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1152 }
```

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}`).

```
1153 \cs_new_protected:Npn \@@_cell_begin:
1154 {
```

`\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.

```
1155   \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).

```
1156   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1157   \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.

```
1158   \int_compare:nNnT \c@jCol = \c_one_int
1159     { \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:`.

```
1160   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1161   \@@_tuning_not_tabular_begin:
1162   \@@_tuning_first_row:
1163   \@@_tuning_last_row:
1164   \g_@@_row_style_tl
1165 }
```

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.

```

1166 \cs_new_protected:Npn \@@_tuning_first_row:
1167 {
1168   \if_int_compare:w \c@iRow = \c_zero_int
1169     \if_int_compare:w \c@jCol > \c_zero_int
1170       \l_@@_code_for_first_row_tl
1171       \xglobal \colorlet { nicematrix-first-row } { . }
1172     \fi:
1173   \fi:
1174 }

```

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.

```

1175 \cs_new_protected:Npn \@@_tuning_last_row:
1176 {
1177   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1178     \l_@@_code_for_last_row_tl
1179     \xglobal \colorlet { nicematrix-last-row } { . }
1180   \fi:
1181 }

```

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

```

1182 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1183 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1184 {
1185   \c_math_toggle_token

```

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

```

1186   \@@_tuning_key_small:
1187 }
1188 \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.

```

1189 \cs_new_protected:Npn \@@_begin_of_row:
1190 {
1191   \int_gincr:N \c@iRow
1192   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim

```

```

1193 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1194 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1195 \pgfpicture
1196 \pgfrememberpicturepositiononpagetrue
1197 \pgfcoordinate
1198   { \@@_env: - row - \int_use:N \c@iRow - base }
1199   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1200 \str_if_empty:NF \l_@@_name_str
1201   {
1202     \pgfnodealias
1203       { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1204       { \@@_env: - row - \int_use:N \c@iRow - base }
1205   }
1206 \endpgfpicture
1207 }

```

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.

```

1208 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1209   {
1210     \int_if_zero:nTF \c@iRow
1211     {
1212       \dim_compare:nNnT { \box_dp:N \l_@@_cell_box } > \g_@@_dp_row_zero_dim
1213       { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1214       \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1215       { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1216     }
1217     {
1218       \int_compare:nNnT \c@iRow = \c_one_int
1219       {
1220         \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1221         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1222       }
1223     }
1224   }
1225 \cs_new_protected:Npn \@@_rotate_cell_box:
1226   {
1227     \box_rotate:Nn \l_@@_cell_box { 90 }
1228     \bool_if:NTF \g_@@_rotate_c_bool
1229     {
1230       \hbox_set:Nn \l_@@_cell_box
1231       {
1232         \c_math_toggle_token
1233         \vcenter { \box_use:N \l_@@_cell_box }
1234         \c_math_toggle_token
1235       }
1236     }
1237     {
1238       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1239       {
1240         \vbox_set_top:Nn \l_@@_cell_box
1241         {
1242           \vbox_to_zero:n { }
1243           \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1244           \box_use:N \l_@@_cell_box
1245         }
1246       }
1247     }
1248     \bool_gset_false:N \g_@@_rotate_bool

```

```

1249   \bool_gset_false:N \g_@@_rotate_c_bool
1250 }
1251 \cs_new_protected:Npn \@@_adjust_size_box:
1252 {
1253   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1254   {
1255     \box_set_wd:Nn \l_@@_cell_box
1256     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1257     \dim_gzero:N \g_@@_blocks_wd_dim
1258   }
1259   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1260   {
1261     \box_set_dp:Nn \l_@@_cell_box
1262     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1263     \dim_gzero:N \g_@@_blocks_dp_dim
1264   }
1265   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1266   {
1267     \box_set_ht:Nn \l_@@_cell_box
1268     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1269     \dim_gzero:N \g_@@_blocks_ht_dim
1270   }
1271 }
1272 \cs_new_protected:Npn \@@_cell_end:
1273 {

```

The following command is nullified in the tabulars.

```

1274   \@@_tuning_not_tabular_end:
1275   \hbox_set_end:
1276   \@@_cell_end_i:
1277 }
1278 \cs_new_protected:Npn \@@_cell_end_i:
1279 {

```

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.

```

1280   \g_@@_cell_after_hook_tl
1281   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1282   \@@_adjust_size_box:
1283   \box_set_ht:Nn \l_@@_cell_box
1284   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1285   \box_set_dp:Nn \l_@@_cell_box
1286   { \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”).

```

1287   \@@_update_max_cell_width:

```

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

```

1288   \@@_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.

```

1289   \bool_if:NTF \g_@@_empty_cell_bool
1290     { \box_use_drop:N \l_@@_cell_box }
1291     {
1292       \bool_if:NTF \g_@@_not_empty_cell_bool
1293         \@@_node_for_cell:
1294         {
1295           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1296             \@@_node_for_cell:
1297             { \box_use_drop:N \l_@@_cell_box }
1298         }
1299     }
1300   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1301     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1302   \bool_gset_false:N \g_@@_empty_cell_bool
1303   \bool_gset_false:N \g_@@_not_empty_cell_bool
1304 }

```

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

```

1305 \cs_new_protected:Npn \@@_update_max_cell_width:
1306   {
1307     \dim_gset:Nn \g_@@_max_cell_width_dim
1308     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1309   }

```

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`).

```

1310 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1311   {
1312     \@@_math_toggle:
1313     \hbox_set_end:
1314     \bool_if:NF \g_@@_rotate_bool
1315       {
1316         \hbox_set:Nn \l_@@_cell_box
1317         {
1318           \makebox [ \l_@@_col_width_dim ] [ s ]
1319           { \hbox_unpack_drop:N \l_@@_cell_box }
1320         }
1321       }
1322     \@@_cell_end_i:
1323   }

```

```

1324 \pgfset
1325   {
1326     nicematrix / cell-node /.style =
1327     {
1328       inner~sep = \c_zero_dim ,
1329       minimum~width = \c_zero_dim
1330     }
1331   }

```

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

```

1332 \cs_new_protected:Npn \@@_node_for_cell:
1333 {
1334   \pgfpicture
1335   \pgfsetbaseline \c_zero_dim
1336   \pgfrememberpicturepositiononpagetrue
1337   \pgfset { nicematrix / cell-node }
1338   \pgfnode
1339     { rectangle }
1340     { base }
1341     {

```

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

```

1342     \set@color
1343     \box_use_drop:N \l_@@_cell_box
1344   }
1345   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1346   { \l_@@_pgf_node_code_tl }
1347   \str_if_empty:NF \l_@@_name_str
1348   {
1349     \pgfnodealias
1350       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1351       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1352   }
1353   \endpgfpicture
1354 }

```

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.

```

1355 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1356 {
1357   \cs_new_protected:Npn \@@_patch_node_for_cell:
1358   {
1359     \hbox_set:Nn \l_@@_cell_box
1360     {
1361       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1362       \hbox_overlap_left:n
1363       {
1364         \pgfsys@markposition
1365         { \@@_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.

```

1366         #1
1367       }
1368     \box_use:N \l_@@_cell_box
1369     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1370     \hbox_overlap_left:n
1371     {
1372       \pgfsys@markposition
1373       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1374     }
1375     #1
1376   }
1377 }
1378 }

```

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

```

1379 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1380 {

```



```

1381 \@@_patch_node_for_cell:n
1382   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1383 }
1384 { \@@_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).

```

1385 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1386   {
1387     \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1388     { \g_@@_#2_lines_tl }
1389     {
1390       \use:c { @@_draw_#2 : nnn }
1391       { \int_use:N \c@iRow }
1392       { \int_use:N \c@jCol }
1393       { \exp_not:n { #3 } }
1394     }
1395   }

1396 \cs_generate_variant:Nn \@@_array:n { o }
1397 \cs_new_protected:Npn \@@_array:n
1398   {
1399     % \begin{macrocode}
1400     \dim_set:Nn \col@sep
1401     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1402     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1403     { \cs_set_nopar:Npn \@halignto { } }
1404     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1405 \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. Re-
mark 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).
1406 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1407 }

```

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`.

```

1408 \bool_if:nTF

```

```

1409 { \c_@@_tagging_array_bool && ! \c_@@_revtex_bool }
1410 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1411 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1412 \cs_new_protected:Npn \@@_create_row_node:
1413 {
1414   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1415   {
1416     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1417     \@@_create_row_node_i:
1418   }
1419 }
1420 \cs_new_protected:Npn \@@_create_row_node_i:
1421 {

```

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

```

1422   \hbox
1423   {
1424     \bool_if:NT \l_@@_code_before_bool
1425     {
1426       \vtop
1427       {
1428         \skip_vertical:N 0.5\arrayrulewidth
1429         \pgfsys@markposition
1430         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1431         \skip_vertical:N -0.5\arrayrulewidth
1432       }
1433     }
1434     \pgfpicture
1435     \pgfrememberpicturepositiononpagetrue
1436     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1437     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1438     \str_if_empty:NF \l_@@_name_str
1439     {
1440       \pgfnodealias
1441       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1442       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1443     }
1444     \endpgfpicture
1445   }
1446 }

```

```

1447 \cs_new_protected:Npn \@@_everycr:
1448 {
1449   \bool_if:NT \c_@@_testphase_table_bool
1450   {
1451     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1452     \tbl_update_cell_data_for_next_row:
1453   }
1454   \int_gzero:N \c@jCol
1455   \bool_gset_false:N \g_@@_after_col_zero_bool
1456   \bool_if:NF \g_@@_row_of_col_done_bool
1457   {
1458     \@@_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).

```

1459     \clist_if_empty:NF \l_@@_hlines_clist
1460     {
1461       \str_if_eq:eeF \l_@@_hlines_clist { all }
1462       {
1463         \clist_if_in:NeT

```

```

1464         \l_@@_hlines_clist
1465         { \int_eval:n { \c@iRow + 1 } }
1466     }
1467     {

```

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.

```

1468         \int_compare:nNnT \c@iRow > { -1 }
1469         {
1470             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1471             { \hrule height \arrayrulewidth width \c_zero_dim }
1472         }
1473     }
1474 }
1475 }
1476 }

```

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

```

1477 \cs_set_protected:Npn \@@_renew_dots:
1478 {
1479     \cs_set_eq:NN \ldots \@@_Ldots
1480     \cs_set_eq:NN \cdots \@@_Cdots
1481     \cs_set_eq:NN \vdots \@@_Vdots
1482     \cs_set_eq:NN \ddots \@@_Ddots
1483     \cs_set_eq:NN \iddots \@@_Iddots
1484     \cs_set_eq:NN \dots \@@_Ldots
1485     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1486 }
1487 \cs_new_protected:Npn \@@_test_color_inside:
1488 {
1489     \bool_if:NF \l_@@_color_inside_bool
1490     {

```

We will issue an error only during the first run.

```

1491         \bool_if:NF \g_@@_aux_found_bool
1492         { \@@_error:n { without~color~inside } }
1493     }
1494 }

```

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

```

1495 \hook_gput_code:nnn { begindocument } { . }
1496 {
1497     \IfPackageLoadedTF { colortbl }
1498     {
1499         \cs_set_protected:Npn \@@_redefine_everycr:
1500         { \CT@everycr { \noalign { \@@_everycr: } } }
1501     }
1502     {
1503         \cs_new_protected:Npn \@@_redefine_everycr:
1504         { \everycr { \noalign { \@@_everycr: } } }
1505     }
1506 }

```

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

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

```

1507 \hook_gput_code:nnn { begindocument } { . }
1508 {
1509   \IfPackageLoadedTF { booktabs }
1510   {
1511     \cs_new_protected:Npn \@@_patch_booktabs:
1512       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1513   }
1514   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1515 }

```

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

```

1516 \cs_new_protected:Npn \@@_some_initialization:
1517 {
1518   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1519   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1520   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1521   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1522   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1523   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1524 }

```

```

1525 \cs_new_protected:Npn \@@_pre_array_ii:
1526 {

```

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

```

1527   \int_gzero:N \g_@@_total_X_weight_int
1528   \@@_expand_clist:N \l_@@_hlines_clist
1529   \@@_expand_clist:N \l_@@_vlines_clist
1530   \@@_patch_booktabs:
1531   \box_clear_new:N \l_@@_cell_box
1532   \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}`).

```

1533   \bool_if:NT \l_@@_small_bool
1534   {
1535     \cs_set_nopar:Npn \arraystretch { 0.47 }
1536     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1537     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1538   }

```

```

1539   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1540   {
1541     \tl_put_right:Nn \@@_begin_of_row:
1542     {
1543       \pgfsys@markposition
1544       { \@@_env: - row - \int_use:N \c@iRow - base }
1545     }
1546   }

```

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

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

```

1547   \bool_if:nTF
1548     { \c_@@_tagging_array_bool && ! \c_@@_revtex_bool }
1549     {
1550       \cs_set_nopar:Npn \ar@ialign
1551         {
1552           \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1553           \@@_redefine_everycr:
1554           \dim_zero:N \tabskip
1555           \@@_some_initialization:

```

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

```

1556           \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1557           \halign
1558         }
1559     }

```

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

```

1560   {
1561     \cs_set_nopar:Npn \ialign
1562     {
1563       \@@_redefine_everycr:
1564       \dim_zero:N \tabskip
1565       \@@_some_initialization:
1566       \cs_set_eq:NN \ialign \@@_old_ialign:
1567       \halign
1568     }
1569   }

```

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).

```

1570   \cs_set_eq:NN \@@_old_ldots \ldots
1571   \cs_set_eq:NN \@@_old_cdots \cdots
1572   \cs_set_eq:NN \@@_old_vdots \vdots
1573   \cs_set_eq:NN \@@_old_ddots \ddots
1574   \cs_set_eq:NN \@@_old_iddots \iddots
1575   \bool_if:NTF \l_@@_standard_cline_bool
1576     { \cs_set_eq:NN \cline \@@_standard_cline }
1577     { \cs_set_eq:NN \cline \@@_cline }
1578   \cs_set_eq:NN \Ldots \@@_Ldots
1579   \cs_set_eq:NN \Cdots \@@_Cdots
1580   \cs_set_eq:NN \Vdots \@@_Vdots
1581   \cs_set_eq:NN \Ddots \@@_Ddots
1582   \cs_set_eq:NN \Iddots \@@_Iddots
1583   \cs_set_eq:NN \Hline \@@_Hline:
1584   \cs_set_eq:NN \Hspace \@@_Hspace:
1585   \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1586   \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1587   \cs_set_eq:NN \Block \@@_Block:
1588   \cs_set_eq:NN \rotate \@@_rotate:
1589   \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1590   \cs_set_eq:NN \dotfill \@@_dotfill:
1591   \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1592   \cs_set_eq:NN \diagbox \@@_diagbox:nn
1593   \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1594   \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1595   \seq_map_inline:Nn \l_@@_custom_line_commands_seq

```

```

1596     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1597 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1598 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1599 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1600 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1601 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1602   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1603 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1604   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1605 \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:`.

```

1606 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1607 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1608   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1609 \@@_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!).

```

1610 \tl_if_exist:NT \l_@@_note_in_caption_tl
1611   {
1612     \tl_if_empty:NF \l_@@_note_in_caption_tl
1613       {
1614         \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1615         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1616       }
1617   }

```

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).

```

1618 \seq_gclear:N \g_@@_multicolumn_cells_seq
1619 \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).

```

1620 \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`).

```

1621 \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.

```

1622 \int_gzero_new:N \g_@@_col_total_int
1623 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1624 \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.

```

1625 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1626 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1627 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1628 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1629 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1630 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

```

```

1631 \tl_gclear:N \g_nicematrix_code_before_tl
1632 \tl_gclear:N \g_@@_pre_code_before_tl
1633 }

```

This is the end of \@@_pre_array_ii:.

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

```

1634 \cs_new_protected:Npn \@@_pre_array:
1635 {
1636   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1637   \int_gzero_new:N \c@iRow
1638   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1639   \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).

```

1640 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1641 {
1642   \bool_set_true:N \l_@@_last_row_without_value_bool
1643   \bool_if:NT \g_@@_aux_found_bool
1644     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1645 }
1646 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1647 {
1648   \bool_if:NT \g_@@_aux_found_bool
1649     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1650 }

```

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”.

```

1651 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1652 {
1653   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1654     {
1655       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1656         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1657       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1658         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1659     }
1660 }

```

```

1661 \seq_gclear:N \g_@@_cols_vlism_seq
1662 \seq_gclear:N \g_@@_submatrix_seq

```

Now the \CodeBefore.

```

1663 \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.

```

1664 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1665 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1666 \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.

```
1667 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1668 \@@_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.

```
1669 \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]`).

```
1670 \dim_zero_new:N \l_@@_left_delim_dim
1671 \dim_zero_new:N \l_@@_right_delim_dim
1672 \bool_if:NTF \g_@@_delims_bool
1673 {
```

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

```
1674 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1675 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1676 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1677 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1678 }
1679 {
1680 \dim_gset:Nn \l_@@_left_delim_dim
1681 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1682 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1683 }
```

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).

```
1684 \hbox_set:Nw \l_@@_the_array_box
1685 \bool_if:NT \c_@@_testphase_table_bool
1686 { \UseTaggingSocket { tbl / hmode / begin } }
1687 \skip_horizontal:N \l_@@_left_margin_dim
1688 \skip_horizontal:N \l_@@_extra_left_margin_dim
1689 \c_math_toggle_token
1690 \bool_if:NTF \l_@@_light_syntax_bool
1691 { \use:c { @@-light-syntax } }
1692 { \use:c { @@-normal-syntax } }
1693 }
```

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

```
1694 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1695 {
1696 \tl_set:Nn \l_tmpa_tl { #1 }
1697 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1698 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1699 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1700 \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...

```
1701   \@@_pre_array:
1702   }
```

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).

```
1703 \cs_new_protected:Npn \@@_pre_code_before:
1704   {
```

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).

```
1705   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1706   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1707   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1708   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

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

```
1709   \pgfsys@markposition { \@@_env: - position }
1710   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1711   \pgfpicture
1712   \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1713   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1714   {
1715     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1716     \pgfcoordinate { \@@_env: - row - ##1 }
1717     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1718   }
```

Now, the recreation of the `col` nodes.

```
1719   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1720   {
1721     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1722     \pgfcoordinate { \@@_env: - col - ##1 }
1723     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1724   }
```

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

```
1725   \@@_create_diag_nodes:
```

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

```
1726   \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1727   \endpgfpicture
```

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

```
1728   \@@_create_blocks_nodes:
```

```

1729 \IfPackageLoadedT { tikz }
1730 {
1731   \tikzset
1732   {
1733     every-picture / .style =
1734     { overlay , name-prefix = \@@_env: - }
1735   }
1736 }
1737 \cs_set_eq:NN \cellcolor \@@_cellcolor
1738 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1739 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1740 \cs_set_eq:NN \rowcolor \@@_rowcolor
1741 \cs_set_eq:NN \rowcolors \@@_rowcolors
1742 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1743 \cs_set_eq:NN \arraycolor \@@_arraycolor
1744 \cs_set_eq:NN \columncolor \@@_columncolor
1745 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1746 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1747 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1748 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1749 }

```

```

1750 \cs_new_protected:Npn \@@_exec_code_before:
1751 {

```

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...

```

1752 \clist_map_inline:Nn \l_@@_corners_cells_clist
1753 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1754 \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`.

```

1755 \@@_add_to_colors_seq:nn { { nocolor } } { }
1756 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1757 \group_begin:

```

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

```

1758 \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`).

```

1759 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1760 { \@@_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`.

```

1761 \exp_last_unbraced:No \@@_CodeBefore_keys:
1762 \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.

```

1763 \@@_actually_color:
1764 \l_@@_code_before_tl
1765 \q_stop

```

```

1766 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1767 \group_end:
1768 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1769 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1770 }

```

```

1771 \keys_define:nn { nicematrix / CodeBefore }
1772 {
1773   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1774   create-cell-nodes .default:n = true ,
1775   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1776   sub-matrix .value_required:n = true ,
1777   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1778   delimiters / color .value_required:n = true ,
1779   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1780 }
1781 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1782 {
1783   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1784   \@@_CodeBefore:w
1785 }

```

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.

```

1786 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1787 {
1788   \bool_if:NT \g_@@_aux_found_bool
1789   {
1790     \@@_pre_code_before:
1791     #1
1792   }
1793 }

```

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.

```

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

```

```

1818     }
1819 \int_step_inline:nn \c@iRow
1820 {
1821     \pgfnodealias
1822     { \@@_env: - ##1 - last }
1823     { \@@_env: - ##1 - \int_use:N \c@jCol }
1824 }
1825 \int_step_inline:nn \c@jCol
1826 {
1827     \pgfnodealias
1828     { \@@_env: - last - ##1 }
1829     { \@@_env: - \int_use:N \c@iRow - ##1 }
1830 }
1831 \@@_create_extra_nodes:
1832 }

```

```

1833 \cs_new_protected:Npn \@@_create_blocks_nodes:
1834 {
1835     \pgfpicture
1836     \pgf@relevantforpicturesizefalse
1837     \pgfrememberpicturepositiononpagetrue
1838     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1839     { \@@_create_one_block_node:nnnnn ##1 }
1840     \endpgfpicture
1841 }

```

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

```

1842 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1843 {
1844     \tl_if_empty:nF { #5 }
1845     {
1846         \@@_qpoint:n { col - #2 }
1847         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1848         \@@_qpoint:n { #1 }
1849         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1850         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1851         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1852         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1853         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1854         \@@_pgf_rect_node:nnnnn
1855         { \@@_env: - #5 }
1856         { \dim_use:N \l_tmpa_dim }
1857         { \dim_use:N \l_tmpb_dim }
1858         { \dim_use:N \l_@@_tmpc_dim }
1859         { \dim_use:N \l_@@_tmpd_dim }
1860     }
1861 }

```

```

1862 \cs_new_protected:Npn \@@_patch_for_revtext:
1863 {
1864     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1865     \cs_set_eq:NN \@array \@array@array
1866     \cs_set_eq:NN \@tabular \@tabular@array
1867     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1868     \cs_set_eq:NN \array \array@array
1869     \cs_set_eq:NN \endarray \endarray@array
1870     \cs_set:Npn \endtabular { \endarray $\egroup} % $

```

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

```

1871 \cs_set_eq:NN \@mkpream \@mkpream@array
1872 \cs_set_eq:NN \@classx \@classx@array
1873 \cs_set_eq:NN \insert@column \insert@column@array
1874 \cs_set_eq:NN \@arraycr \@arraycr@array
1875 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1876 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1877 }

```

10 The environment `{NiceArrayWithDelims}`

```

1878 \NewDocumentEnvironment { NiceArrayWithDelims }
1879 { m m 0 { } m ! 0 { } t \CodeBefore }
1880 {
1881 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revteX:
1882 \@@_provide_pgfsyspdfmark:
1883 \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 exponent to a matrix in a mathematical formula.

```

1884 \bgroup

1885 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1886 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1887 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1888 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1889 \int_gzero:N \g_@@_block_box_int
1890 \dim_zero:N \g_@@_width_last_col_dim
1891 \dim_zero:N \g_@@_width_first_col_dim
1892 \bool_gset_false:N \g_@@_row_of_col_done_bool
1893 \str_if_empty:NT \g_@@_name_env_str
1894 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1895 \bool_if:NTF \l_@@_tabular_bool
1896 \mode_leave_vertical:
1897 \@@_test_if_math_mode:
1898 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet-in-env } }
1899 \bool_set_true:N \l_@@_in_env_bool

```

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

```

1900 \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.

```

1901 \cs_if_exist:NT \tikz@library@external@loaded
1902 {
1903 \tikzexternaldisable
1904 \cs_if_exist:NT \ifstandalone
1905 { \tikzset { external / optimize = false } }
1906 }

```

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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1907 \int_gincr:N \g_@@_env_int
1908 \bool_if:NF \l_@@_block_auto_columns_width_bool
1909 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1910 \seq_gclear:N \g_@@_blocks_seq
1911 \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`.

```

1912 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1913 \seq_gclear:N \g_@@_pos_of_xdots_seq
1914 \tl_gclear_new:N \g_@@_code_before_tl
1915 \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.

```

1916 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1917 {
1918   \bool_gset_true:N \g_@@_aux_found_bool
1919   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1920 }
1921 { \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.

```

1922 \tl_gclear:N \g_@@_aux_tl
1923 \tl_if_empty:NF \g_@@_code_before_tl
1924 {
1925   \bool_set_true:N \l_@@_code_before_bool
1926   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1927 }
1928 \tl_if_empty:NF \g_@@_pre_code_before_tl
1929 { \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`.

```

1930 \bool_if:NTF \g_@@_delims_bool
1931 { \keys_set:nn { nicematrix / pNiceArray } }
1932 { \keys_set:nn { nicematrix / NiceArray } }
1933 { #3 , #5 }

```

```

1934 \@@_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:.`

```

1935 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1936 }

```

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

```

1937 {
1938   \bool_if:NTF \l_@@_light_syntax_bool
1939   { \use:c { end @@-light-syntax } }
1940   { \use:c { end @@-normal-syntax } }
1941   \c_math_toggle_token
1942   \skip_horizontal:N \l_@@_right_margin_dim
1943   \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1944
1945 % awful workaround
1946 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1947 {
1948   \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1949   {
1950     \skip_horizontal:N - \l_@@_columns_width_dim
1951     \bool_if:NTF \l_@@_tabular_bool
1952       { \skip_horizontal:n { - 2 \tabcolsep } }
1953       { \skip_horizontal:n { - 2 \arraycolsep } }
1954   }
1955 }
1956 \hbox_set_end:

```

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

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

```

1957 \bool_if:NT \l_@@_width_used_bool
1958 {
1959   \int_if_zero:nT \g_@@_total_X_weight_int
1960   { \@@_error_or_warning:n { width-without-X-columns } }
1961 }

```

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 .

```

1962 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1963 {
1964   \tl_gput_right:Ne \g_@@_aux_tl
1965   {
1966     \bool_set_true:N \l_@@_X_columns_aux_bool
1967     \dim_set:Nn \l_@@_X_columns_dim
1968     {
1969       \dim_compare:nNnTF
1970       {
1971         \dim_abs:n
1972         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1973       }
1974       <
1975       { 0.001 pt }
1976       { \dim_use:N \l_@@_X_columns_dim }
1977       {
1978         \dim_eval:n
1979         {
1980           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1981           / \int_use:N \g_@@_total_X_weight_int
1982           + \l_@@_X_columns_dim
1983         }
1984       }
1985     }
1986   }
1987 }

```

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).

```

1988 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1989 {
1990   \bool_if:NF \l_@@_last_row_without_value_bool
1991   {
1992     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1993     {
1994       \@@_error:n { Wrong-last-row }
1995       \int_gset_eq:NN \l_@@_last_row_int \c_iRow

```

```

1996         }
1997     }
1998 }

```

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

```

1999     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2000     \bool_if:NTF \g_@@_last_col_found_bool
2001     { \int_gdecr:N \c@jCol }
2002     {
2003         \int_compare:nNnT \l_@@_last_col_int > { -1 }
2004         { \@@_error:n { last~col~not~used } }
2005     }

```

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

```

2006     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2007     \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 89).

```

2008     \int_if_zero:nT \l_@@_first_col_int
2009     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2010     \bool_if:nTF { ! \g_@@_delims_bool }
2011     {
2012         \str_if_eq:eeTF \l_@@_baseline_tl { c }
2013         \@@_use_arraybox_with_notes_c:
2014         {
2015             \str_if_eq:eeTF \l_@@_baseline_tl { b }
2016             \@@_use_arraybox_with_notes_b:
2017             \@@_use_arraybox_with_notes:
2018         }
2019     }

```

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).

```

2020     {
2021         \int_if_zero:nTF \l_@@_first_row_int
2022         {
2023             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2024             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2025         }
2026         { \dim_zero:N \l_tmpa_dim }

```

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

```

2027         \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2028         {
2029             \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2030             \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2031         }
2032         { \dim_zero:N \l_tmpb_dim }
2033     \hbox_set:Nn \l_tmpa_box
2034     {
2035         \c_math_toggle_token
2036         \@@_color:o \l_@@_delimiters_color_tl
2037         \exp_after:wN \left \g_@@_left_delim_tl
2038         \vcenter

```

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

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

2039

{

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.

```

2040     \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2041     \hbox
2042     {
2043         \bool_if:NTF \l_@@_tabular_bool
2044             { \skip_horizontal:N -\tabcolsep }
2045             { \skip_horizontal:N -\arraycolsep }
2046         \@@_use_arraybox_with_notes_c:
2047         \bool_if:NTF \l_@@_tabular_bool
2048             { \skip_horizontal:N -\tabcolsep }
2049             { \skip_horizontal:N -\arraycolsep }
2050     }

```

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

```

2051     \skip_vertical:N -\l_tmpb_dim
2052     \skip_vertical:N \arrayrulewidth
2053 }
2054 \exp_after:wN \right \g_@@_right_delim_tl
2055 \c_math_toggle_token
2056 }

```

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.

```

2057     \bool_if:NTF \l_@@_delimiters_max_width_bool
2058     {
2059         \@@_put_box_in_flow_bis:nn
2060         \g_@@_left_delim_tl
2061         \g_@@_right_delim_tl
2062     }
2063     \@@_put_box_in_flow:
2064 }

```

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

```

2065     \bool_if:NT \g_@@_last_col_found_bool
2066     { \skip_horizontal:N \g_@@_width_last_col_dim }
2067     \bool_if:NT \l_@@_preamble_bool
2068     {
2069         \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2070         { \@@_warning_gredirect_none:n { columns-not-used } }
2071     }
2072     \@@_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 exponent to a matrix in a mathematical formula.

```

2073     \egroup

```

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

```

2074     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2075     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2076     \iow_now:Ne \@mainaux
2077     {
2078         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2079         { \exp_not:o \g_@@_aux_tl }
2080     }
2081     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2082     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2083 }

```

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

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.

```

2084 \cs_new_protected:Npn \@@_transform_preamble:
2085   {
2086     \@@_transform_preamble_i:
2087     \@@_transform_preamble_ii:
2088   }

2089 \cs_new_protected:Npn \@@_transform_preamble_i:
2090   {
2091     \int_gzero:N \c@jCol

```

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

```

2092   \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.

```

2093   \bool_gset_false:N \g_tmpb_bool

```

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

```

2094   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2095   \int_zero:N \l_tmpa_int
2096   \tl_gclear:N \g_@@_array_preamble_tl
2097   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2098   {
2099     \tl_gset:Nn \g_@@_array_preamble_tl
2100     { ! { \skip_horizontal:N \arrayrulewidth } }
2101   }
2102   {
2103     \clist_if_in:NnT \l_@@_vlines_clist 1
2104     {
2105       \tl_gset:Nn \g_@@_array_preamble_tl
2106       { ! { \skip_horizontal:N \arrayrulewidth } }
2107     }
2108   }

```

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

```

2109   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2110   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2111   \@@_replace_columncolor:
2112   }

```

```

2113 \hook_gput_code:nnn { begindocument } { . }
2114 {
2115   \IfPackageLoadedTF { colortbl }
2116   {

```

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).

```

2117     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2118     \cs_new_protected:Npn \@@_replace_columncolor:
2119     {
2120         \regex_replace_all:NnN
2121         \c_@@_columncolor_regex
2122         { \c { @@_columncolor_preamble } }
2123         \g_@@_array_preamble_tl
2124     }
2125 }
2126 {
2127     \cs_new_protected:Npn \@@_replace_columncolor:
2128     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2129 }
2130 }

```

```

2131 \cs_new_protected:Npn \@@_transform_preamble_ii:
2132 {

```

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

```

2133     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2134     {
2135         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2136         { \bool_gset_true:N \g_@@_delims_bool }
2137     }
2138     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2139     \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).

```

2140     \int_if_zero:nTF \l_@@_first_col_int
2141     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2142     {
2143         \bool_if:NF \g_@@_delims_bool
2144         {
2145             \bool_if:NF \l_@@_tabular_bool
2146             {
2147                 \clist_if_empty:NT \l_@@_vlines_clist
2148                 {
2149                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2150                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2151                 }
2152             }
2153         }
2154     }
2155     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2156     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2157     {
2158         \bool_if:NF \g_@@_delims_bool
2159         {
2160             \bool_if:NF \l_@@_tabular_bool
2161             {
2162                 \clist_if_empty:NT \l_@@_vlines_clist
2163                 {
2164                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2165                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2166                 }
2167             }
2168         }

```

```

2167     }
2168   }
2169 }

```

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`).

```

2170   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2171   {
2172     \tl_gput_right:Nn \g_@@_array_preamble_tl
2173     { > { \@@_error_too_much_cols: } l }
2174   }
2175 }

```

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.

```

2176 \cs_new_protected:Npn \@@_rec_preamble:n #1
2177 {

```

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

```

2178   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2179   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2180   {

```

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

```

2181     \cs_if_exist:cTF { NC @ find @ #1 }
2182     {
2183       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2184       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2185     }
2186     {
2187       \str_if_eq:nnTF { #1 } { S }
2188       { \@@_fatal:n { unknown~column~type~S } }
2189       { \@@_fatal:nn { unknown~column~type } { #1 } }
2190     }
2191   }
2192 }

```

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

```

2193 \cs_new_protected:Npn \@@_c #1
2194 {
2195   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2196   \tl_gclear:N \g_@@_pre_cell_tl
2197   \tl_gput_right:Nn \g_@@_array_preamble_tl
2198   { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2199   \int_gincr:N \c@jCol
2200   \@@_rec_preamble_after_col:n
2201 }

2202 \cs_new_protected:Npn \@@_l #1
2203 {
2204   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2205   \tl_gclear:N \g_@@_pre_cell_tl
2206   \tl_gput_right:Nn \g_@@_array_preamble_tl

```

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

```

2207     {
2208     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2209     l
2210     < \@@_cell_end:
2211     }
2212     \int_gincr:N \c@jCol
2213     \@@_rec_preamble_after_col:n
2214     }
2215 \cs_new_protected:Npn \@@_r #1
2216 {
2217     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2218     \tl_gclear:N \g_@@_pre_cell_tl
2219     \tl_gput_right:Nn \g_@@_array_preamble_tl
2220     {
2221     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2222     r
2223     < \@@_cell_end:
2224     }
2225     \int_gincr:N \c@jCol
2226     \@@_rec_preamble_after_col:n
2227     }

```

For ! and @

```

2228 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2229 {
2230     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2231     \@@_rec_preamble:n
2232     }
2233 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2234 \cs_new_protected:cpn { @@ _ | } #1
2235 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2236     \int_incr:N \l_tmpa_int
2237     \@@_make_preamble_i_i:n
2238     }
2239 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2240 {
2241     \str_if_eq:nnTF { #1 } { | }
2242     { \use:c { @@ _ | } | }
2243     { \@@_make_preamble_i_ii:nn { } #1 }
2244     }
2245 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2246 {
2247     \str_if_eq:nnTF { #2 } { [ ]
2248     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2249     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2250     }
2251 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2252 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2253 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2254 {
2255     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2256     \tl_gput_right:Ne \g_@@_array_preamble_tl
2257     {

```

Here, the command \dim_use:N is mandatory.

```

2258     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2259     }
2260     \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

2261 {
2262   \@@_vline:n
2263   {
2264     position = \int_eval:n { \c@jCol + 1 } ,
2265     multiplicity = \int_use:N \l_tmpa_int ,
2266     total-width = \dim_use:N \l_@@_rule_width_dim ,
2267     #2
2268   }

```

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.

```

2269   }
2270   \int_zero:N \l_tmpa_int
2271   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2272   \@@_rec_preamble:n #1
2273 }

```

```

2274 \cs_new_protected:cpn { @@_ > } #1 #2
2275 {
2276   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2277   \@@_rec_preamble:n
2278 }

```

```

2279 \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.

```

2280 \keys_define:nn { nicematrix / p-column }
2281 {
2282   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2283   r .value_forbidden:n = true ,
2284   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2285   c .value_forbidden:n = true ,
2286   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2287   l .value_forbidden:n = true ,
2288   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2289   S .value_forbidden:n = true ,
2290   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2291   p .value_forbidden:n = true ,
2292   t .meta:n = p ,
2293   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2294   m .value_forbidden:n = true ,
2295   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2296   b .value_forbidden:n = true
2297 }

```

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

```

2298 \cs_new_protected:Npn \@@_p #1
2299 {
2300   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2301   \@@_make_preamble_ii_i:n
2302 }
2303 \cs_set_eq:NN \@@_b \@@_p
2304 \cs_set_eq:NN \@@_m \@@_p
2305 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2306 {
2307   \str_if_eq:nnTF { #1 } { [ ]
2308     { \@@_make_preamble_ii_ii:w [ ]
2309       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2310     }

```

```

2311 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2312 { \@@_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.

```

2313 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2314 {

```

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).

```

2315     \str_set:Nn \l_@@_hpos_col_str { j }
2316     \@@_keys_p_column:n { #1 }
2317     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2318 }

2319 \cs_new_protected:Npn \@@_keys_p_column:n #1
2320 { \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.

```

2321 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2322 {
2323     \use:e
2324     {
2325         \@@_make_preamble_ii_v:nnnnnnn
2326         { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2327         { \dim_eval:n { #1 } }
2328     }

```

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.

```

2329         \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2330         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2331     }

```

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

```

2332         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2333         { \str_lowercase:o \l_@@_hpos_col_str }
2334     }
2335     \IfPackageLoadedTF { ragged2e }
2336     {
2337         \str_case:on \l_@@_hpos_col_str
2338         {
2339             c { \exp_not:N \Centering }
2340             l { \exp_not:N \RaggedRight }
2341             r { \exp_not:N \RaggedLeft }
2342         }
2343     }
2344     {
2345         \str_case:on \l_@@_hpos_col_str
2346         {
2347             c { \exp_not:N \centering }
2348             l { \exp_not:N \raggedright }
2349             r { \exp_not:N \raggedleft }
2350         }
2351     }
2352     #3
2353 }
2354 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2355 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2356 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2357 { #2 }

```

```

2358     {
2359         \str_case:onF \l_@@_hpos_col_str
2360         {
2361             { j } { c }
2362             { si } { c }
2363         }

```

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

```

2364         { \str_lowercase:o \l_@@_hpos_col_str }
2365     }
2366 }

```

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

```

2367     \int_gincr:N \c_jCol
2368     \@@_rec_preamble_after_col:n
2369 }

```

#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*.

```

2370 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2371 {
2372     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2373     {
2374         \tl_gput_right:Nn \g_@@_array_preamble_tl
2375         { > \@@_test_if_empty_for_S: }
2376     }
2377     {
2378         \tl_gput_right:Nn \g_@@_array_preamble_tl
2379         { > \@@_test_if_empty: }
2380     }
2381     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2382     \tl_gclear:N \g_@@_pre_cell_tl
2383     \tl_gput_right:Nn \g_@@_array_preamble_tl
2384     {
2385         > {

```

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.

```

2386         \dim_set:Nn \l_@@_col_width_dim { #2 }
2387         \bool_if:NT \c_@@_testphase_table_bool
2388         { \tag_struct_begin:n { tag = Div } }
2389         \@@_cell_begin:

```

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

```

2390     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2391     \everypar
2392     {
2393         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2394         \everypar { }
2395     }
2396     \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.).

```
2397         #3
```

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

```
2398         \g_@@_row_style_tl
2399         \arraybackslash
2400         #5
2401     }
2402     #8
2403     < {
2404         #6
```

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

```
2405         \@finalstrut \@arstrutbox
2406         \use:c { end #7 }
```

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

```
2407         #4
2408         \@@_cell_end:
2409         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2410     }
2411 }
2412 }
```

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

```
2413 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2414 {
```

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 `&`).

```
2415     \group_align_safe_begin:
2416     \peek_meaning:NTF &
2417     {
2418         \group_align_safe_end:
2419         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2420         {
```

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

```
2421         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2422         \skip_horizontal:N \l_@@_col_width_dim
2423     }
2424 }
2425 { \group_align_safe_end: }
2426 }

2427 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2428 {
2429     \peek_meaning:NT \__siunitx_table_skip:n
2430     { \bool_gset_true:N \g_@@_empty_cell_bool }
2431 }
```

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

```
2432 \cs_new_protected:Npn \@@_center_cell_box:
2433 {
```

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

```

2434 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2435 {
2436   \int_compare:nNnT
2437     { \box_ht:N \l_@@_cell_box }
2438     >

```

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).

```

2439   { \box_ht:N \strutbox }
2440   {
2441     \hbox_set:Nn \l_@@_cell_box
2442     {
2443       \box_move_down:nn
2444       {
2445         ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2446           + \baselineskip ) / 2
2447       }
2448       { \box_use:N \l_@@_cell_box }
2449     }
2450   }
2451 }
2452 }

```

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

```

2453 \cs_new_protected:Npn \@@_V #1 #2
2454 {
2455   \str_if_eq:nnTF { #1 } { [ ]
2456     { \@@_make_preamble_V_i:w [ ]
2457       { \@@_make_preamble_V_i:w [ ] { #2 } }
2458     }
2459   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2460     { \@@_make_preamble_V_ii:nn { #1 } }
2461   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2462     {
2463       \str_set:Nn \l_@@_vpos_col_str { p }
2464       \str_set:Nn \l_@@_hpos_col_str { j }
2465       \@@_keys_p_column:n { #1 }
2466       \IfPackageLoadedTF { varwidth }
2467         { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2468         {
2469           \@@_error_or_warning:n { varwidth-not-loaded }
2470           \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2471         }
2472     }

```

For w and W

```

2473 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2474 \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.

```

2475 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2476 {
2477   \str_if_eq:nnTF { #3 } { s }
2478     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2479     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2480 }

```

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.

```

2481 \cs_new_protected:Npn \@@_make_preamble_w_i:nmmm #1 #2
2482 {
2483   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2484   \tl_gclear:N \g_@@_pre_cell_tl
2485   \tl_gput_right:Nn \g_@@_array_preamble_tl
2486     {
2487     > {
2488       \dim_set:Nn \l_@@_col_width_dim { #2 }
2489       \@@_cell_begin:
2490       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2491     }
2492     c
2493     < {
2494       \@@_cell_end_for_w_s:
2495       #1
2496       \@@_adjust_size_box:
2497       \box_use_drop:N \l_@@_cell_box
2498     }
2499   }
2500   \int_gincr:N \c@jCol
2501   \@@_rec_preamble_after_col:n
2502 }

```

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

```

2503 \cs_new_protected:Npn \@@_make_preamble_w_ii:nmmm #1 #2 #3 #4
2504 {
2505   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2506   \tl_gclear:N \g_@@_pre_cell_tl
2507   \tl_gput_right:Nn \g_@@_array_preamble_tl
2508     {
2509     > {

```

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.

```

2510       \dim_set:Nn \l_@@_col_width_dim { #4 }
2511       \hbox_set:Nw \l_@@_cell_box
2512       \@@_cell_begin:
2513       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2514     }
2515     c
2516     < {
2517       \@@_cell_end:
2518       \hbox_set_end:
2519       #1
2520       \@@_adjust_size_box:
2521       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2522     }
2523   }

```

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

```

2524   \int_gincr:N \c@jCol
2525   \@@_rec_preamble_after_col:n
2526 }

2527 \cs_new_protected:Npn \@@_special_W:
2528 {
2529   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2530     { \@@_warning:n { W~warning } }
2531 }

```

For S (of siunitx).

```

2532 \cs_new_protected:Npn \@@_S #1 #2
2533 {
2534   \str_if_eq:nnTF { #2 } { [ ]
2535     { \@@_make_preamble_S:w [ ]
2536       { \@@_make_preamble_S:w [ ] { #2 } }
2537     }
2538 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2539 { \@@_make_preamble_S_i:n { #1 } }
2540 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2541 {
2542   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2543   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2544   \tl_gc_clear:N \g_@@_pre_cell_tl
2545   \tl_gput_right:Nn \g_@@_array_preamble_tl
2546     {
2547     > {
2548       \@@_cell_begin:
2549       \keys_set:nn { siunitx } { #1 }
2550       \siunitx_cell_begin:w
2551     }
2552     c
2553     < { \siunitx_cell_end: \@@_cell_end: }
2554   }

```

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

```

2555   \int_gincr:N \c@jCol
2556   \@@_rec_preamble_after_col:n
2557 }

```

For (, [and \{.

```

2558 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2559 {
2560   \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.

```

2561   \int_if_zero:nTF \c@jCol
2562   {
2563     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2564     {

```

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

```

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

```

```

2585     \@@_rec_preamble:n
2586   }
2587   { \@@_rec_preamble:n #2 }
2588 }

```

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

```

2589 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2590 { \use:c { @@ _ \token_to_str:N ( ) } }

```

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

```

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

```

```

2641     \@@_rec_preamble:n #3
2642   }
2643 }

2644 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2645 { \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 @{...}.

```

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

2671 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2672 {
2673   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2674   \@@_rec_preamble_after_col:n
2675 }

```

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.

```

2676 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2677 {
2678   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2679   {
2680     \tl_gput_right:Nn \g_@@_array_preamble_tl
2681     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2682   }
2683   {
2684     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2685     {
2686       \tl_gput_right:Nn \g_@@_array_preamble_tl
2687       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2688     }
2689     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2690   }
2691   \@@_rec_preamble:n
2692 }

```

```

2693 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2694 {
2695   \tl_clear:N \l_tmpa_tl
2696   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2697   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2698 }

```

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.

```

2699 \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`.

```

2700 \cs_new_protected:Npn \@@_X #1 #2
2701 {
2702   \str_if_eq:nnTF { #2 } { [ ]
2703     { \@@_make_preamble_X:w [ ] }
2704     { \@@_make_preamble_X:w [ ] #2 }
2705   }
2706   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2707     { \@@_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`).

```

2708 \keys_define:nn { nicematrix / X-column }
2709   { 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`.

```

2710 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2711 {

```

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`).

```

2712   \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`).

```

2713   \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`.

```

2714   \int_zero_new:N \l_@@_weight_int
2715   \int_set_eq:NN \l_@@_weight_int \c_one_int
2716   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2717   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2718   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2719     {
2720       \@@_error_or_warning:n { negative-weight }
2721       \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2722     }
2723   \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).

```

2724 \bool_if:NTF \l_@@_X_columns_aux_bool
2725 {
2726   \@@_make_preamble_ii_iv:nnn
2727   { \l_@@_weight_int \l_@@_X_columns_dim }
2728   { minipage }
2729   { \@@_no_update_width: }
2730 }
2731 {
2732   \tl_gput_right:Nn \g_@@_array_preamble_tl
2733   {
2734     > {
2735       \@@_cell_begin:
2736       \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`.

```

2737 \NotEmpty

```

The following code will nullify the box of the cell.

```

2738 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2739 { \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`.

```

2740 \begin { minipage } { 5 cm } \arraybackslash
2741 }
2742 c
2743 < {
2744   \end { minipage }
2745   \@@_cell_end:
2746 }
2747 }
2748 \int_gincr:N \c@jCol
2749 \@@_rec_preamble_after_col:n
2750 }
2751 }

```

```

2752 \cs_new_protected:Npn \@@_no_update_width:
2753 {
2754   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2755   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2756 }

```

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

```

2757 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2758 {
2759   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2760   { \int_eval:n { \c@jCol + 1 } }
2761   \tl_gput_right:Ne \g_@@_array_preamble_tl
2762   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2763   \@@_rec_preamble:n
2764 }

```

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.

```

2765 \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).

```

2766 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2767   { \@@_fatal:n { Preamble-forgotten } }
2768 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2769 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2770 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \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).

```

2771 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2772   {

```

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

```

2773   \multispan { #1 }
2774   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2775   \begingroup
2776   \bool_if:NT \c_@@_testphase_table_bool
2777     { \tbl_update_multicolumn_cell_data:n { #1 } }
2778   \cs_set_nopar:Npn \@addamp
2779     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2780   \tl_gclear:N \g_@@_preamble_tl
2781   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2782   \exp_args:No \@mkpream \g_@@_preamble_tl
2783   \@addtopreamble \@empty
2784   \endgroup
2785   \bool_if:NT \c_@@_testphase_table_bool
2786     { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2787   \int_compare:nNnT { #1 } > \c_one_int
2788   {
2789     \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2790       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2791     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2792     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2793       {
2794         {
2795           \int_if_zero:nTF \c@jCol
2796             { \int_eval:n { \c@iRow + 1 } }
2797             { \int_use:N \c@iRow }
2798         }
2799         { \int_eval:n { \c@jCol + 1 } }
2800         {
2801           \int_if_zero:nTF \c@jCol
2802             { \int_eval:n { \c@iRow + 1 } }
2803             { \int_use:N \c@iRow }
2804         }
2805         { \int_eval:n { \c@jCol + #1 } }
2806         { } % for the name of the block

```

```

2807     }
2808 }

```

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

```

2809 \RenewDocumentCommand \cellcolor { 0 { } m }
2810 {
2811   \@@_test_color_inside:
2812   \tl_gput_right:N\g_@@_pre_code_before_tl
2813   {
2814     \@@_rectanglecolor [ ##1 ]
2815     { \exp_not:n { ##2 } }
2816     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2817     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2818   }
2819   \ignorespaces
2820 }

```

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

```

2821 \cs_set_nopar:Npn \@sharp { #3 }
2822 \@arstrut
2823 \@preamble
2824 \null

```

We add some lines.

```

2825 \int_gadd:Nn \c@jCol { #1 - 1 }
2826 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2827 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2828 \ignorespaces
2829 }

```

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`.

```

2830 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2831 {
2832   \str_case:nnF { #1 }
2833   {
2834     c { \@@_make_m_preamble_i:n #1 }
2835     l { \@@_make_m_preamble_i:n #1 }
2836     r { \@@_make_m_preamble_i:n #1 }
2837     > { \@@_make_m_preamble_ii:nn #1 }
2838     ! { \@@_make_m_preamble_ii:nn #1 }
2839     @ { \@@_make_m_preamble_ii:nn #1 }
2840     | { \@@_make_m_preamble_iii:n #1 }
2841     p { \@@_make_m_preamble_iv:nnn t #1 }
2842     m { \@@_make_m_preamble_iv:nnn c #1 }
2843     b { \@@_make_m_preamble_iv:nnn b #1 }
2844     w { \@@_make_m_preamble_v:nnnn { } #1 }
2845     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2846     \q_stop { }
2847   }
2848   {
2849     \cs_if_exist:cTF { NC @ find @ #1 }
2850     {
2851       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2852       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2853     }
2854     {
2855       \str_if_eq:nnTF { #1 } { S }
2856       { \@@_fatal:n { unknown~column~type~S } }
2857       { \@@_fatal:nn { unknown~column~type } { #1 } }

```

```

2858     }
2859   }
2860 }

```

For c, l and r

```

2861 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2862 {
2863   \tl_gput_right:Nn \g_@@_preamble_tl
2864   {
2865     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2866     #1
2867     < \@@_cell_end:
2868   }

```

We test for the presence of a <.

```

2869   \@@_make_m_preamble_x:n
2870 }

```

For >, ! and @

```

2871 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2872 {
2873   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2874   \@@_make_m_preamble:n
2875 }

```

For |

```

2876 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2877 {
2878   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2879   \@@_make_m_preamble:n
2880 }

```

For p, m and b

```

2881 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2882 {
2883   \tl_gput_right:Nn \g_@@_preamble_tl
2884   {
2885     > {
2886       \@@_cell_begin:
2887       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2888       \mode_leave_vertical:
2889       \arraybackslash
2890       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2891     }
2892     c
2893     < {
2894       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2895       \end { minipage }
2896       \@@_cell_end:
2897     }
2898   }

```

We test for the presence of a <.

```

2899   \@@_make_m_preamble_x:n
2900 }

```

For w and W

```

2901 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2902 {
2903   \tl_gput_right:Nn \g_@@_preamble_tl
2904   {
2905     > {
2906       \dim_set:Nn \l_@@_col_width_dim { #4 }
2907       \hbox_set:Nw \l_@@_cell_box

```

```

2908     \@@_cell_begin:
2909     \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2910   }
2911   c
2912   < {
2913     \@@_cell_end:
2914     \hbox_set_end:
2915     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2916     #1
2917     \@@_adjust_size_box:
2918     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2919   }
2920 }

```

We test for the presence of a <.

```

2921 \@@_make_m_preamble_x:n
2922 }

```

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

```

2923 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2924 {
2925   \str_if_eq:nnTF { #1 } { < }
2926   \@@_make_m_preamble_ix:n
2927   { \@@_make_m_preamble:n { #1 } }
2928 }
2929 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2930 {
2931   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2932   \@@_make_m_preamble_x:n
2933 }

```

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).

```

2934 \cs_new_protected:Npn \@@_put_box_in_flow:
2935 {
2936   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2937   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2938   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2939   { \box_use_drop:N \l_tmpa_box }
2940   \@@_put_box_in_flow_i:
2941 }

```

The command \@@_put_box_in_flow_i: is used when the value of \l_@@_baseline_tl is different of c (which is the initial value and the most used).

```

2942 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2943 {
2944   \pgfpicture
2945     \@@_qpoint:n { row - 1 }
2946     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2947     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2948     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2949     \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).

```

2950   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2951   {
2952     \int_set:Nn \l_tmpa_int
2953     {
2954       \str_range:Nnn

```

```

2955         \l_@@_baseline_tl
2956         6
2957         { \tl_count:o \l_@@_baseline_tl }
2958     }
2959     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2960 }
2961 {
2962     \str_if_eq:eeTF \l_@@_baseline_tl { t }
2963     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2964     {
2965         \str_if_eq:onTF \l_@@_baseline_tl { b }
2966         { \int_set_eq:NN \l_tmpa_int \c@iRow }
2967         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2968     }
2969     \bool_lazy_or:nnT
2970     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2971     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2972     {
2973         \@@_error:n { bad-value-for-baseline }
2974         \int_set_eq:NN \l_tmpa_int \c_one_int
2975     }
2976     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2977         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2978     }
2979     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2980     \endpgfpicture
2981     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2982     \box_use_drop:N \l_tmpa_box
2983 }

```

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).

```

2984 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2985 {

```

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.

```

2986     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2987     {
2988         \int_compare:nNnT \c@jCol > \c_one_int
2989         {
2990             \box_set_wd:Nn \l_@@_the_array_box
2991             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2992         }
2993     }

```

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

```

2994     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2995     \bool_if:NT \l_@@_caption_above_bool
2996     {
2997         \tl_if_empty:NF \l_@@_caption_tl
2998         {
2999             \bool_set_false:N \g_@@_caption_finished_bool
3000             \int_gzero:N \c@tabularnote
3001             \@@_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).

```

3002     \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3003     {
3004         \tl_gput_right:Ne \g_@@_aux_tl
3005         {
3006             \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3007             { \int_use:N \g_@@_notes_caption_int }
3008         }
3009         \int_gzero:N \g_@@_notes_caption_int
3010     }
3011 }
3012 }

```

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

```

3013     \hbox
3014     {
3015         \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.

```

3016     \@@_create_extra_nodes:
3017     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3018 }

```

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).

```

3019     \bool_lazy_any:nT
3020     {
3021         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3022         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3023         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3024     }
3025     \@@_insert_tabularnotes:
3026     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3027     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3028     \end { minipage }
3029 }

```

```

3030 \cs_new_protected:Npn \@@_insert_caption:
3031 {
3032     \tl_if_empty:NF \l_@@_caption_tl
3033     {
3034         \cs_if_exist:NTF \c@caption
3035         { \@@_insert_caption_i: }
3036         { \@@_error:n { caption~outside~float } }
3037     }
3038 }

```

```

3039 \cs_new_protected:Npn \@@_insert_caption_i:
3040 {
3041     \group_begin:

```

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

```

3042     \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.

```

3043   \IfPackageLoadedT { floatrow }
3044   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3045   \tl_if_empty:NTF \l_@@_short_caption_tl
3046   { \caption }
3047   { \caption [ \l_@@_short_caption_tl ] }
3048   { \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.

```

3049   \bool_if:NF \g_@@_caption_finished_bool
3050   {
3051     \bool_gset_true:N \g_@@_caption_finished_bool
3052     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3053     \int_gzero:N \c@tabularnote
3054   }
3055   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3056   \group_end:
3057 }

3058 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3059 {
3060   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3061   \@@_gredirect_none:n { tabularnote-below-the-tabular }
3062 }

3063 \cs_new_protected:Npn \@@_insert_tabularnotes:
3064 {
3065   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3066   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3067   \skip_vertical:N 0.65ex

```

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

```

3068   \group_begin:
3069   \l_@@_notes_code_before_tl
3070   \tl_if_empty:NF \g_@@_tabularnote_tl
3071   {
3072     \g_@@_tabularnote_tl \par
3073     \tl_gclear:N \g_@@_tabularnote_tl
3074   }

```

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.

```

3075   \int_compare:nNnT \c@tabularnote > \c_zero_int
3076   {
3077     \bool_if:NTF \l_@@_notes_para_bool
3078     {
3079       \begin { tabularnotes* }
3080         \seq_map_inline:Nn \g_@@_notes_seq
3081         { \@@_one_tabularnote:nm ##1 }
3082         \strut
3083         \end { tabularnotes* }

```

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

```

3084     \par
3085   }
3086   {
3087     \tabularnotes
3088     \seq_map_inline:Nn \g_@@_notes_seq

```

```

3089         { \@@_one_tabularnote:nn #1 }
3090         \strut
3091     \endtabularnotes
3092 }
3093 }
3094 \unskip
3095 \group_end:
3096 \bool_if:NT \l_@@_notes_bottomrule_bool
3097 {
3098     \IfPackageLoadedTF { booktabs }
3099     {

```

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

```

3100         \skip_vertical:N \aboverulesep

```

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

```

3101         { \CT@arc@ \hrule height \heavyrulewidth }
3102     }
3103     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3104 }
3105 \l_@@_notes_code_after_tl
3106 \seq_gclear:N \g_@@_notes_seq
3107 \seq_gclear:N \g_@@_notes_in_caption_seq
3108 \int_gzero:N \c@tabularnote
3109 }

```

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

```

3110 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3111 {
3112     \tl_if_novalue:nTF { #1 }
3113     { \item }
3114     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3115 }

```

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`.

```

3116 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3117 {
3118     \pgfpicture
3119     \@@_qpoint:n { row - 1 }
3120     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3121     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3122     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3123     \endpgfpicture
3124     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3125     \int_if_zero:nT \l_@@_first_row_int
3126     {
3127         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3128         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3129     }
3130     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3131 }

```

Now, the general case.

```

3132 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3133 {

```

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

```

3134     \str_if_eq:eeT \l_@@_baseline_tl { t }
3135     { \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`.

```

3136 \pgfpicture
3137 \@@_qpoint:n { row - 1 }
3138 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3139 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3140 {
3141   \int_set:Nn \l_tmpa_int
3142   {
3143     \str_range:Nnn
3144       \l_@@_baseline_tl
3145       6
3146     { \tl_count:o \l_@@_baseline_tl }
3147   }
3148   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3149 }
3150 {
3151   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3152   \bool_lazy_or:nnT
3153     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3154     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3155     {
3156       \@@_error:n { bad~value~for~baseline }
3157       \int_set:Nn \l_tmpa_int 1
3158     }
3159   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3160 }
3161 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3162 \endpgfpicture
3163 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3164 \int_if_zero:nT \l_@@_first_row_int
3165 {
3166   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3167   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3168 }
3169 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3170 }

```

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.

```

3171 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3172 {

```

We will compute the real width of both delimiters used.

```

3173   \dim_zero_new:N \l_@@_real_left_delim_dim
3174   \dim_zero_new:N \l_@@_real_right_delim_dim
3175   \hbox_set:Nn \l_tmpb_box
3176   {
3177     \c_math_toggle_token
3178     \left #1
3179     \vcenter
3180     {
3181       \vbox_to_ht:nn
3182         { \box_ht_plus_dp:N \l_tmpa_box }
3183         { }
3184     }
3185     \right .
3186     \c_math_toggle_token
3187   }
3188   \dim_set:Nn \l_@@_real_left_delim_dim
3189     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3190   \hbox_set:Nn \l_tmpb_box

```

```

3191     {
3192       \c_math_toggle_token
3193       \left .
3194       \vbox_to_ht:nn
3195         { \box_ht_plus_dp:N \l_tmpa_box }
3196         { }
3197       \right #2
3198       \c_math_toggle_token
3199     }
3200   \dim_set:Nn \l_@@_real_right_delim_dim
3201     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3202   \skip_horizontal:N \l_@@_left_delim_dim
3203   \skip_horizontal:N -\l_@@_real_left_delim_dim
3204   \@@_put_box_in_flow:
3205   \skip_horizontal:N \l_@@_right_delim_dim
3206   \skip_horizontal:N -\l_@@_real_right_delim_dim
3207 }

```

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).

```

3208 \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).

```

3209 {
3210   \peek_remove_spaces:n
3211   {
3212     \peek_meaning:NTF \end
3213     \@@_analyze_end:Nn
3214     {
3215       \@@_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`).

```

3216       \@@_array:o \g_@@_array_preamble_tl
3217     }
3218   }
3219 }
3220 {
3221   \@@_create_col_nodes:
3222   \endarray
3223 }

```

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

```

3224 \NewDocumentEnvironment { @@-light-syntax } { b }
3225 {

```

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`.

```

3226   \tl_if_empty:nT { #1 }
3227     { \@@_fatal:n { empty-environment } }
3228   \tl_if_in:nnT { #1 } { & }
3229     { \@@_fatal:n { ampersand-in-light-syntax } }
3230   \tl_if_in:nnT { #1 } { \ }
3231     { \@@_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`.

```
3232   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3233   }
```

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.

```
3234   {
3235     \@@_create_col_nodes:
3236     \endarray
3237   }

3238   \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3239   {
3240     \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).

```
3241     \seq_clear_new:N \l_@@_rows_seq
```

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

```
3242     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3243     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3244       \seq_set_split:Nee
3245       \seq_set_split:Non
3246       \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3247     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3248     \tl_if_empty:NF \l_tmpa_tl
3249     { \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.

```
3250     \int_compare:nNnT \l_@@_last_row_int = { -1 }
3251     { \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`.

```
3252     \tl_build_begin:N \l_@@_new_body_tl
3253     \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3254     \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3255     \@@_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).

```
3256     \seq_map_inline:Nn \l_@@_rows_seq
3257     {
3258       \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3259       \@@_line_with_light_syntax:n { ##1 }
3260     }
3261     \tl_build_end:N \l_@@_new_body_tl

3262     \int_compare:nNnT \l_@@_last_col_int = { -1 }
3263     {
3264       \int_set:Nn \l_@@_last_col_int
3265       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3266     }
```

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.

```
3267 \@@_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`).

```
3268 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3269 }
3270 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3271 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3272 {
3273   \seq_clear_new:N \l_@@_cells_seq
3274   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3275   \int_set:Nn \l_@@_nb_cols_int
3276   {
3277     \int_max:nn
3278     \l_@@_nb_cols_int
3279     { \seq_count:N \l_@@_cells_seq }
3280   }
3281   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3282   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3283   \seq_map_inline:Nn \l_@@_cells_seq
3284   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3285 }
```

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`.

```
3286 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3287 {
3288   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3289   { \@@_fatal:n { empty-environment } }
```

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

```
3290 \end { #2 }
3291 }
```

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`).

```
3292 \cs_new:Npn \@@_create_col_nodes:
3293 {
3294   \crrc
3295   \int_if_zero:nT \l_@@_first_col_int
3296   {
3297     \omit
3298     \hbox_overlap_left:n
3299     {
3300       \bool_if:NT \l_@@_code_before_bool
3301       { \pgfsys@markposition { \@@_env: - col - 0 } }
3302       \pgfpicture
3303       \pgfrememberpicturepositiononpagetrue
3304       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3305       \str_if_empty:NF \l_@@_name_str
3306       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3307       \endpgfpicture
3308       \skip_horizontal:N 2\col@sep
3309       \skip_horizontal:N \g_@@_width_first_col_dim
3310     }
3311 }
```

```

3311     &
3312     }
3313     \omit

```

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

```

3314     \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`).

```

3315     \int_if_zero:nTF \l_@@_first_col_int
3316     {
3317         \bool_if:NT \l_@@_code_before_bool
3318         {
3319             \hbox
3320             {
3321                 \skip_horizontal:N -0.5\arrayrulewidth
3322                 \pgfsys@markposition { \@@_env: - col - 1 }
3323                 \skip_horizontal:N 0.5\arrayrulewidth
3324             }
3325         }
3326         \pgfpicture
3327         \pgfrememberpicturepositiononpagetrue
3328         \pgfcoordinate { \@@_env: - col - 1 }
3329         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3330         \str_if_empty:NF \l_@@_name_str
3331         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3332         \endpgfpicture
3333     }
3334     {
3335         \bool_if:NT \l_@@_code_before_bool
3336         {
3337             \hbox
3338             {
3339                 \skip_horizontal:N 0.5\arrayrulewidth
3340                 \pgfsys@markposition { \@@_env: - col - 1 }
3341                 \skip_horizontal:N -0.5\arrayrulewidth
3342             }
3343         }
3344         \pgfpicture
3345         \pgfrememberpicturepositiononpagetrue
3346         \pgfcoordinate { \@@_env: - col - 1 }
3347         { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3348         \str_if_empty:NF \l_@@_name_str
3349         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3350         \endpgfpicture
3351     }

```

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.

```

3352     \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3353     \bool_if:NF \l_@@_auto_columns_width_bool
3354     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3355     {
3356         \bool_lazy_and:nnTF
3357         \l_@@_auto_columns_width_bool
3358         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3359         { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3360         { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3361         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3362     }

```

```

3363 \skip_horizontal:N \g_tmpa_skip
3364 \hbox
3365 {
3366   \bool_if:NT \l_@@_code_before_bool
3367   {
3368     \hbox
3369     {
3370       \skip_horizontal:N -0.5\arrayrulewidth
3371       \pgfsys@markposition { \@@_env: - col - 2 }
3372       \skip_horizontal:N 0.5\arrayrulewidth
3373     }
3374   }
3375   \pgfpicture
3376   \pgfrememberpicturepositiononpagetrue
3377   \pgfcoordinate { \@@_env: - col - 2 }
3378   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3379   \str_if_empty:NF \l_@@_name_str
3380   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3381   \endpgfpicture
3382 }

```

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.

```

3383 \int_gset_eq:NN \g_tmpa_int \c_one_int
3384 \bool_if:NTF \g_@@_last_col_found_bool
3385 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3386 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3387 {
3388   &
3389   \omit
3390   \int_gincr:N \g_tmpa_int

```

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

```

3391 \skip_horizontal:N \g_tmpa_skip
3392 \bool_if:NT \l_@@_code_before_bool
3393 {
3394   \hbox
3395   {
3396     \skip_horizontal:N -0.5\arrayrulewidth
3397     \pgfsys@markposition
3398     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3399     \skip_horizontal:N 0.5\arrayrulewidth
3400   }
3401 }

```

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

```

3402 \pgfpicture
3403 \pgfrememberpicturepositiononpagetrue
3404 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3405 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3406 \str_if_empty:NF \l_@@_name_str
3407 {
3408   \pgfnodealias
3409   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3410   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3411 }
3412 \endpgfpicture
3413 }

3414 &
3415 \omit

```

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

```

3416 \int_if_zero:nT \g_@@_col_total_int
3417 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3418 \skip_horizontal:N \g_tmpa_skip
3419 \int_gincr:N \g_tmpa_int
3420 \bool_lazy_any:nF
3421 {
3422   \g_@@_delims_bool
3423   \l_@@_tabular_bool
3424   { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3425   \l_@@_exterior_arraycolsep_bool
3426   \l_@@_bar_at_end_of_pream_bool
3427 }
3428 { \skip_horizontal:N -\col@sep }
3429 \bool_if:NT \l_@@_code_before_bool
3430 {
3431   \hbox
3432   {
3433     \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.

```

3434   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3435   { \skip_horizontal:N -\arraycolsep }
3436   \pgfsys@markposition
3437   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3438   \skip_horizontal:N 0.5\arrayrulewidth
3439   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3440   { \skip_horizontal:N \arraycolsep }
3441 }
3442 }
3443 \pgfpicture
3444 \pgfrememberpicturepositiononpagetrue
3445 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3446 {
3447   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3448   {
3449     \pgfpoint
3450     { - 0.5 \arrayrulewidth - \arraycolsep }
3451     \c_zero_dim
3452   }
3453   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3454 }
3455 \str_if_empty:NF \l_@@_name_str
3456 {
3457   \pgfnodealias
3458   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3459   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3460 }
3461 \endpgfpicture

3462 \bool_if:NT \g_@@_last_col_found_bool
3463 {
3464   \hbox_overlap_right:n
3465   {
3466     \skip_horizontal:N \g_@@_width_last_col_dim
3467     \skip_horizontal:N \col@sep
3468     \bool_if:NT \l_@@_code_before_bool
3469     {
3470       \pgfsys@markposition
3471       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3472     }
3473   \pgfpicture

```

```

3474     \pgfrememberpicturepositiononpagetrue
3475     \pgfcoordinate
3476     { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3477     \pgfpointorigin
3478     \str_if_empty:NF \l_@@_name_str
3479     {
3480         \pgfnodealias
3481         {
3482             \l_@@_name_str - col
3483             - \int_eval:n { \g_@@_col_total_int + 1 }
3484         }
3485         { \l_@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3486     }
3487     \endpgfpicture
3488 }
3489 }
3490 % \cr
3491 }

```

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

```

3492 \tl_const:Nn \c_@@_preamble_first_col_tl
3493 {
3494     >
3495     {

```

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

```

3496     \cs_set_eq:NN \CodeAfter \l_@@_CodeAfter_i:
3497     \bool_gset_true:N \g_@@_after_col_zero_bool
3498     \l_@@_begin_of_row:
3499     \hbox_set:Nw \l_@@_cell_box
3500     \l_@@_math_toggle:
3501     \l_@@_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”.

```

3502     \int_compare:nNnT \c_iRow > \c_zero_int
3503     {
3504         \bool_lazy_or:nnT
3505         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3506         { \int_compare_p:nNn \c_iRow < \l_@@_last_row_int }
3507         {
3508             \l_@@_code_for_first_col_tl
3509             \xglobal \colorlet { nicematrix-first-col } { . }
3510         }
3511     }
3512 }

```

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`.

```

3513     l
3514     <
3515     {
3516         \l_@@_math_toggle:
3517         \hbox_set_end:
3518         \bool_if:NT \g_@@_rotate_bool \l_@@_rotate_cell_box:
3519         \l_@@_adjust_size_box:
3520         \l_@@_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.

```

3521     \dim_gset:Nn \g_@@_width_first_col_dim
3522     { \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.

```

3523     \hbox_overlap_left:n
3524     {
3525         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3526         \@@_node_for_cell:
3527         { \box_use_drop:N \l_@@_cell_box }
3528         \skip_horizontal:N \l_@@_left_delim_dim
3529         \skip_horizontal:N \l_@@_left_margin_dim
3530         \skip_horizontal:N \l_@@_extra_left_margin_dim
3531     }
3532     \bool_gset_false:N \g_@@_empty_cell_bool
3533     \skip_horizontal:N -2\col@sep
3534 }
3535 }

```

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

```

3536 \tl_const:Nn \c_@@_preamble_last_col_tl
3537 {
3538     >
3539     {
3540         \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).

```

3541     \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.

```

3542     \bool_gset_true:N \g_@@_last_col_found_bool
3543     \int_gincr:N \c@jCol
3544     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3545     \hbox_set:Nw \l_@@_cell_box
3546     \@@_math_toggle:
3547     \@@_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”.

```

3548     \int_compare:nNnT \c@iRow > \c_zero_int
3549     {
3550         \bool_lazy_or:nnT
3551         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3552         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3553         {
3554             \l_@@_code_for_last_col_tl
3555             \xglobal \colorlet { nicematrix-last-col } { . }
3556         }
3557     }
3558 }
3559 l
3560 <
3561 {
3562     \@@_math_toggle:
3563     \hbox_set_end:
3564     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3565     \@@_adjust_size_box:
3566     \@@_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.

```

3567     \dim_gset:Nn \g_@@_width_last_col_dim
3568     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3569     \skip_horizontal:N -2\col@sep

```

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

```

3570     \hbox_overlap_right:n
3571     {

```

```

3572         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3573         {
3574             \skip_horizontal:N \l_@@_right_delim_dim
3575             \skip_horizontal:N \l_@@_right_margin_dim
3576             \skip_horizontal:N \l_@@_extra_right_margin_dim
3577             \@@_node_for_cell:
3578         }
3579     }
3580     \bool_gset_false:N \g_@@_empty_cell_bool
3581 }
3582 }

```

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

```

3583 \NewDocumentEnvironment { NiceArray } { }
3584 {
3585     \bool_gset_false:N \g_@@_delims_bool
3586     \str_if_empty:NT \g_@@_name_env_str
3587     { \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).

```

3588     \NiceArrayWithDelims . .
3589 }
3590 { \endNiceArrayWithDelims }

```

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

```

3591 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3592 {
3593     \NewDocumentEnvironment { #1 NiceArray } { }
3594     {
3595         \bool_gset_true:N \g_@@_delims_bool
3596         \str_if_empty:NT \g_@@_name_env_str
3597         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3598         \@@_test_if_math_mode:
3599         \NiceArrayWithDelims #2 #3
3600     }
3601     { \endNiceArrayWithDelims }
3602 }
3603 \@@_def_env:nnn p ( )
3604 \@@_def_env:nnn b [ ]
3605 \@@_def_env:nnn B \{ \}
3606 \@@_def_env:nnn v | |
3607 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3608 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3609 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3610 {
3611     \bool_set_false:N \l_@@_preamble_bool
3612     \tl_clear:N \l_tmpa_tl
3613     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3614     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3615     \tl_put_right:Nn \l_tmpa_tl
3616     {
3617         *

```

```

3618     {
3619         \int_case:nnF \l_@@_last_col_int
3620         {
3621             { -2 } { \c@MaxMatrixCols }
3622             { -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).

```

3623     }
3624     { \int_eval:n { \l_@@_last_col_int - 1 } }
3625 }
3626 { #2 }
3627 }
3628 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3629 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3630 }
3631 \clist_map_inline:nn { p , b , B , v , V }
3632 {
3633     \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3634     {
3635         \bool_gset_true:N \g_@@_delims_bool
3636         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3637         \int_if_zero:nT \l_@@_last_col_int
3638         {
3639             \bool_set_true:N \l_@@_last_col_without_value_bool
3640             \int_set:Nn \l_@@_last_col_int { -1 }
3641         }
3642         \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3643         \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3644     }
3645     { \use:c { end #1 NiceArray } }
3646 }

```

We define also an environment {NiceMatrix}

```

3647 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3648 {
3649     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3650     \int_if_zero:nT \l_@@_last_col_int
3651     {
3652         \bool_set_true:N \l_@@_last_col_without_value_bool
3653         \int_set:Nn \l_@@_last_col_int { -1 }
3654     }
3655     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3656     \bool_lazy_or:nnT
3657     { \clist_if_empty_p:N \l_@@_vlines_clist }
3658     { \l_@@_except_borders_bool }
3659     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3660     \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3661 }
3662 { \endNiceArray }

```

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

```

3663 \cs_new_protected:Npn \@@_NotEmpty:
3664 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3665 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3666 {

```

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.

```

3667 \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3668   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3669 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3670 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3671 \tl_if_empty:NF \l_@@_short_caption_tl
3672   {
3673     \tl_if_empty:NT \l_@@_caption_tl
3674       {
3675         \@@_error_or_warning:n { short-caption-without~caption }
3676         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3677       }
3678     }
3679 \tl_if_empty:NF \l_@@_label_tl
3680   {
3681     \tl_if_empty:NT \l_@@_caption_tl
3682     { \@@_error_or_warning:n { label~without~caption } }
3683   }
3684 \NewDocumentEnvironment { TabularNote } { b }
3685   {
3686     \bool_if:NTF \l_@@_in_code_after_bool
3687     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3688     {
3689       \tl_if_empty:NF \g_@@_tabularnote_tl
3690       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3691       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3692     }
3693   }
3694   { }
3695 \@@_settings_for_tabular:
3696 \NiceArray { #2 }
3697 }
3698 {
3699   \endNiceArray
3700   \bool_if:NT \c_@@_testphase_table_bool
3701   { \UseTaggingSocket { tbl / hmode / end } }
3702 }
3703 \cs_new_protected:Npn \@@_settings_for_tabular:
3704   {
3705     \bool_set_true:N \l_@@_tabular_bool
3706     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3707     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3708     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3709   }

3710 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3711   {
3712     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3713     \dim_zero_new:N \l_@@_width_dim
3714     \dim_set:Nn \l_@@_width_dim { #1 }
3715     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3716     \@@_settings_for_tabular:
3717     \NiceArray { #3 }
3718   }
3719   {
3720     \endNiceArray
3721     \int_if_zero:nT \g_@@_total_X_weight_int
3722     { \@@_error:n { NiceTabularX~without~X } }
3723   }

3724 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3725   {
3726     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3727     \dim_set:Nn \l_@@_tabular_width_dim { #1 }

```

```

3728 \keys_set:nm { nicematrix / NiceTabular } { #2 , #4 }
3729 \@@_settings_for_tabular:
3730 \NiceArray { #3 }
3731 }
3732 { \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`).

```

3733 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3734 {
3735   \bool_lazy_all:nT
3736   {
3737     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3738     \l_@@_hvlines_bool
3739     { ! \g_@@_delims_bool }
3740     { ! \l_@@_except_borders_bool }
3741   }
3742   {
3743     \bool_set_true:N \l_@@_except_borders_bool
3744     \clist_if_empty:NF \l_@@_corners_clist
3745     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3746     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3747     {
3748       \@@_stroke_block:nnn
3749       {
3750         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3751         draw = \l_@@_rules_color_tl
3752       }
3753       { 1-1 }
3754       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3755     }
3756   }
3757 }

3758 \cs_new_protected:Npn \@@_after_array:
3759 {

```

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`.

```

3760 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3761 \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.

```

3762 \bool_if:NT \g_@@_last_col_found_bool
3763 { \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`.

```

3764 \bool_if:NT \l_@@_last_col_without_value_bool
3765 { \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.

```

3766   \bool_if:NT \l_@@_last_row_without_value_bool
3767     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3768   \tl_gput_right:Ne \g_@@_aux_tl
3769     {
3770     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3771       {
3772         \int_use:N \l_@@_first_row_int ,
3773         \int_use:N \c@iRow ,
3774         \int_use:N \g_@@_row_total_int ,
3775         \int_use:N \l_@@_first_col_int ,
3776         \int_use:N \c@jCol ,
3777         \int_use:N \g_@@_col_total_int
3778       }
3779     }

```

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`).

```

3780   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3781     {
3782     \tl_gput_right:Ne \g_@@_aux_tl
3783       {
3784         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3785           { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3786       }
3787     }
3788   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3789     {
3790     \tl_gput_right:Ne \g_@@_aux_tl
3791       {
3792         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3793           { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3794         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3795           { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3796       }
3797     }

```

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

```

3798   \@@_create_diag_nodes:

```

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

```

3799   \pgfpicture
3800   \int_step_inline:nn \c@iRow
3801     {
3802     \pgfnodealias
3803       { \@@_env: - ##1 - last }
3804       { \@@_env: - ##1 - \int_use:N \c@jCol }
3805     }
3806   \int_step_inline:nn \c@jCol
3807     {
3808     \pgfnodealias
3809       { \@@_env: - last - ##1 }
3810       { \@@_env: - \int_use:N \c@iRow - ##1 }
3811     }
3812   \str_if_empty:NF \l_@@_name_str
3813     {
3814     \int_step_inline:nn \c@iRow
3815       {
3816       \pgfnodealias
3817         { \l_@@_name_str - ##1 - last }
3818         { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3819     }
3820     \int_step_inline:nm \c@jCol
3821     {
3822         \pgfnodealias
3823         { \l_@@_name_str - last - ##1 }
3824         { \@@_env: - \int_use:N \c@iRow - ##1 }
3825     }
3826 }
3827 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹¹. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

3828     \bool_if:NT \l_@@_parallelize_diags_bool
3829     {
3830         \int_gzero_new:N \g_@@_ddots_int
3831         \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3832         \dim_gzero_new:N \g_@@_delta_x_one_dim
3833         \dim_gzero_new:N \g_@@_delta_y_one_dim
3834         \dim_gzero_new:N \g_@@_delta_x_two_dim
3835         \dim_gzero_new:N \g_@@_delta_y_two_dim
3836     }
3837     \int_zero_new:N \l_@@_initial_i_int
3838     \int_zero_new:N \l_@@_initial_j_int
3839     \int_zero_new:N \l_@@_final_i_int
3840     \int_zero_new:N \l_@@_final_j_int
3841     \bool_set_false:N \l_@@_initial_open_bool
3842     \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.

```

3843     \bool_if:NT \l_@@_small_bool
3844     {
3845         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3846         \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.

```

3847         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3848         { 0.6 \l_@@_xdots_shorten_start_dim }
3849         \dim_set:Nn \l_@@_xdots_shorten_end_dim
3850         { 0.6 \l_@@_xdots_shorten_end_dim }
3851     }

```

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

```

3852     \@@_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.

```

3853     \clist_if_empty:NF \l_@@_corners_clist \@@_compute_corners:

```

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

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

```

3854     \@@_adjust_pos_of_blocks_seq:
3855     \@@_deal_with_rounded_corners:
3856     \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3857     \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3858     \IfPackageLoadedT { tikz }
3859     {
3860         \tikzset
3861         {
3862             every-picture / .style =
3863             {
3864                 overlay ,
3865                 remember-picture ,
3866                 name-prefix = \@@_env: -
3867             }
3868         }
3869     }
3870     \bool_if:NT \c_@@_tagging_array_bool
3871     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3872     \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3873     \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3874     \cs_set_eq:NN \OverBrace \@@_OverBrace
3875     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3876     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3877     \cs_set_eq:NN \line \@@_line
3878     \g_@@_pre_code_after_tl
3879     \tl_gclear:N \g_@@_pre_code_after_tl

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it’s possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That’s why we set `\Code-after` to be *no-op* now.

```

3880     \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).

```

3881     \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`).

```

3882     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3883     { \@@_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:`.

```

3884     \bool_set_true:N \l_@@_in_code_after_bool
3885     \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3886     \scan_stop:
3887     \tl_gclear:N \g_nicematrix_code_after_tl
3888     \group_end:

```

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

```

3889     \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3890     \tl_if_empty:NF \g_@@_pre_code_before_tl
3891     {
3892         \tl_gput_right:Ne \g_@@_aux_tl

```



```

3893     {
3894         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3895         { \exp_not:o \g_@@_pre_code_before_tl }
3896     }
3897     \tl_gclear:N \g_@@_pre_code_before_tl
3898 }
3899 \tl_if_empty:NF \g_nicematrix_code_before_tl
3900 {
3901     \tl_gput_right:Ne \g_@@_aux_tl
3902     {
3903         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3904         { \exp_not:o \g_nicematrix_code_before_tl }
3905     }
3906     \tl_gclear:N \g_nicematrix_code_before_tl
3907 }

3908 \str_gclear:N \g_@@_name_env_str
3909 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹². This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

3910     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3911 }

```

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`.

```

3912 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3913 { \keys_set:nm { 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).

```

3914 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3915 {
3916     \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3917     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3918 }

```

The following command must *not* be protected.

```

3919 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3920 {
3921     { #1 }
3922     { #2 }
3923     {
3924         \int_compare:nNnTF { #3 } > { 99 }
3925         { \int_use:N \c@iRow }
3926         { #3 }
3927     }
3928     {
3929         \int_compare:nNnTF { #4 } > { 99 }
3930         { \int_use:N \c@jCol }
3931         { #4 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3932     }
3933     { #5 }
3934 }

```

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).

```

3935 \hook_gput_code:nnn { begindocument } { . }
3936 {
3937   \cs_new_protected:Npe \@@_draw_dotted_lines:
3938   {
3939     \c_@@_pgfortikzpicture_tl
3940     \@@_draw_dotted_lines_i:
3941     \c_@@_endpgfortikzpicture_tl
3942   }
3943 }

```

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

```

3944 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3945 {
3946   \pgfrememberpicturepositiononpagetrue
3947   \pgf@relevantforpicturesizefalse
3948   \g_@@_HVdotsfor_lines_tl
3949   \g_@@_Vdots_lines_tl
3950   \g_@@_Ddots_lines_tl
3951   \g_@@_Iddots_lines_tl
3952   \g_@@_Cdots_lines_tl
3953   \g_@@_Ldots_lines_tl
3954 }

```

```

3955 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3956 {
3957   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3958   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3959 }

```

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

```

3960 \pgfdeclareshape { @@_diag_node }
3961 {
3962   \savedanchor { \five }
3963   {
3964     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3965     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3966   }
3967   \anchor { 5 } { \five }
3968   \anchor { center } { \pgfpointorigin }
3969   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3970   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3971   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3972   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3973   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
3974   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3975   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3976   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3977   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3978   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3979 }

```


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

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

This command computes:

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

```
4021 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4022 {
```

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

```
4023 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4024 \int_set:Nn \l_@@_initial_i_int { #1 }
4025 \int_set:Nn \l_@@_initial_j_int { #2 }
4026 \int_set:Nn \l_@@_final_i_int { #1 }
4027 \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.

```
4028 \bool_set_false:N \l_@@_stop_loop_bool
4029 \bool_do_until:Nn \l_@@_stop_loop_bool
4030 {
4031   \int_add:Nn \l_@@_final_i_int { #3 }
4032   \int_add:Nn \l_@@_final_j_int { #4 }
4033   \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.

```
4034   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4035     \if_int_compare:w #3 = \c_one_int
4036       \bool_set_true:N \l_@@_final_open_bool
4037     \else:
4038       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4039         \bool_set_true:N \l_@@_final_open_bool
4040       \fi:
4041     \fi:
4042   \else:
4043     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4044       \if_int_compare:w #4 = -1
4045         \bool_set_true:N \l_@@_final_open_bool
4046       \fi:
4047     \else:
4048       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4049         \if_int_compare:w #4 = \c_one_int
4050           \bool_set_true:N \l_@@_final_open_bool
4051         \fi:
4052       \fi:
4053     \fi:
4054   \fi:
```

```
4055     \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.

```
4056     {
```

We do a step backwards.

```
4057         \int_sub:Nn \l_@@_final_i_int { #3 }
4058         \int_sub:Nn \l_@@_final_j_int { #4 }
4059         \bool_set_true:N \l_@@_stop_loop_bool
4060     }
```

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`.

```
4061     {
4062         \cs_if_exist:cTF
4063         {
4064             @@ _ dotted _
4065             \int_use:N \l_@@_final_i_int -
4066             \int_use:N \l_@@_final_j_int
4067         }
4068         {
4069             \int_sub:Nn \l_@@_final_i_int { #3 }
4070             \int_sub:Nn \l_@@_final_j_int { #4 }
4071             \bool_set_true:N \l_@@_final_open_bool
4072             \bool_set_true:N \l_@@_stop_loop_bool
4073         }
4074         {
4075             \cs_if_exist:cTF
4076             {
4077                 pgf @ sh @ ns @ \@@_env:
4078                 - \int_use:N \l_@@_final_i_int
4079                 - \int_use:N \l_@@_final_j_int
4080             }
4081             { \bool_set_true:N \l_@@_stop_loop_bool }
4082         }
4083     }
```

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.

```
4082     {
4083         \cs_set_nopar:cpn
4084         {
4085             @@ _ dotted _
4086             \int_use:N \l_@@_final_i_int -
4087             \int_use:N \l_@@_final_j_int
4088         }
4089         { }
4090     }
4091 }
4092 }
4093 }
```

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

```
4094     \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4095     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4096     \bool_do_until:Nn \l_@@_stop_loop_bool
4097     {
4098         \int_sub:Nn \l_@@_initial_i_int { #3 }
4099         \int_sub:Nn \l_@@_initial_j_int { #4 }
4100         \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.

```

4101     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4102     \if_int_compare:w #3 = \c_one_int
4103         \bool_set_true:N \l_@@_initial_open_bool
4104     \else:
\l_tmpa_int contains \l_@@_col_min_int - 1 (only for efficiency).
4105         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4106         \bool_set_true:N \l_@@_initial_open_bool
4107     \fi:
4108 \fi:
4109 \else:
4110     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4111     \if_int_compare:w #4 = \c_one_int
4112         \bool_set_true:N \l_@@_initial_open_bool
4113     \fi:
4114 \else:
4115     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4116     \if_int_compare:w #4 = -1
4117         \bool_set_true:N \l_@@_initial_open_bool
4118     \fi:
4119 \fi:
4120 \fi:
4121 \fi:
4122 \bool_if:NTF \l_@@_initial_open_bool
4123 {
4124     \int_add:Nn \l_@@_initial_i_int { #3 }
4125     \int_add:Nn \l_@@_initial_j_int { #4 }
4126     \bool_set_true:N \l_@@_stop_loop_bool
4127 }
4128 {
4129     \cs_if_exist:cTF
4130     {
4131         @@ _ dotted _
4132         \int_use:N \l_@@_initial_i_int -
4133         \int_use:N \l_@@_initial_j_int
4134     }
4135     {
4136         \int_add:Nn \l_@@_initial_i_int { #3 }
4137         \int_add:Nn \l_@@_initial_j_int { #4 }
4138         \bool_set_true:N \l_@@_initial_open_bool
4139         \bool_set_true:N \l_@@_stop_loop_bool
4140     }
4141     {
4142         \cs_if_exist:cTF
4143         {
4144             pgf @ sh @ ns @ \@@_env:
4145             - \int_use:N \l_@@_initial_i_int
4146             - \int_use:N \l_@@_initial_j_int
4147         }
4148         { \bool_set_true:N \l_@@_stop_loop_bool }
4149         {
4150             \cs_set_nopar:cpn
4151             {
4152                 @@ _ dotted _
4153                 \int_use:N \l_@@_initial_i_int -
4154                 \int_use:N \l_@@_initial_j_int
4155             }
4156             { }
4157         }
4158     }
4159 }

```

```
4160     }
```

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.

```
4161     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4162     {
4163     { \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`.

```
4164     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4165     { \int_use:N \l_@@_final_i_int }
4166     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4167     { } % for the name of the block
4168     }
4169 }
```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analyse of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```
4170 \cs_new_protected:Npn \@@_open_shorten:
4171 {
4172   \bool_if:NT \l_@@_initial_open_bool
4173   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4174   \bool_if:NT \l_@@_final_open_bool
4175   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4176 }
```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row `#1` and column `#2`. As of now, it’s only the whole array (excepted exterior rows and columns).

```
4177 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4178 {
4179   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4180   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4181   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4182   \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`.

```
4183   \seq_if_empty:NF \g_@@_submatrix_seq
4184   {
4185     \seq_map_inline:Nn \g_@@_submatrix_seq
4186     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4187   }
4188 }
```

`#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:nnnnn #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.

```

4189 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4190 {
4191   \if_int_compare:w #3 > #1
4192   \else:
4193     \if_int_compare:w #1 > #5
4194     \else:
4195       \if_int_compare:w #4 > #2
4196       \else:
4197         \if_int_compare:w #2 > #6
4198         \else:
4199           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4200           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4201           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4202           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4203         \fi:
4204       \fi:
4205     \fi:
4206   \fi:
4207 }

4208 \cs_new_protected:Npn \@@_set_initial_coords:
4209 {
4210   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4211   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4212 }
4213 \cs_new_protected:Npn \@@_set_final_coords:
4214 {
4215   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4216   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4217 }
4218 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4219 {
4220   \pgfpointanchor
4221   {
4222     \@@_env:
4223     - \int_use:N \l_@@_initial_i_int
4224     - \int_use:N \l_@@_initial_j_int
4225   }
4226   { #1 }
4227   \@@_set_initial_coords:
4228 }
4229 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4230 {
4231   \pgfpointanchor
4232   {
4233     \@@_env:
4234     - \int_use:N \l_@@_final_i_int
4235     - \int_use:N \l_@@_final_j_int
4236   }
4237   { #1 }
4238   \@@_set_final_coords:
4239 }

```



```

4240 \cs_new_protected:Npn \@@_open_x_initial_dim:
4241 {
4242   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4243   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4244   {
4245     \cs_if_exist:cT
4246     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4247     {
4248       \pgfpointanchor
4249       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4250       { west }
4251       \dim_set:Nn \l_@@_x_initial_dim
4252       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4253     }
4254   }

```

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

```

4255   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4256   {
4257     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4258     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4259     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4260   }
4261 }

```

```

4262 \cs_new_protected:Npn \@@_open_x_final_dim:
4263 {
4264   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4265   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4266   {
4267     \cs_if_exist:cT
4268     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4269     {
4270       \pgfpointanchor
4271       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4272       { east }
4273       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4274       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4275     }
4276   }

```

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

```

4277   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4278   {
4279     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4280     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4281     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4282   }
4283 }

```

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.

```

4284 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4285 {
4286   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4287   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4288   {
4289     \@@_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.

```

4290   \group_begin:
4291   \@@_open_shorten:

```

```

4292     \int_if_zero:nTF { #1 }
4293     { \color { nicematrix-first-row } }
4294     {

```

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.

```

4295         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4296         { \color { nicematrix-last-row } }
4297     }
4298     \keys_set:nn { nicematrix / xdots } { #3 }
4299     \@@_color:o \l_@@_xdots_color_tl
4300     \@@_actually_draw_Ldots:
4301     \group_end:
4302 }
4303 }

```

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`.

```

4304 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4305 {
4306     \bool_if:NTF \l_@@_initial_open_bool
4307     {
4308         \@@_open_x_initial_dim:
4309         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4310         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4311     }
4312     { \@@_set_initial_coords_from_anchor:n { base-east } }
4313     \bool_if:NTF \l_@@_final_open_bool
4314     {
4315         \@@_open_x_final_dim:
4316         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4317         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4318     }
4319     { \@@_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.

```

4320     \bool_lazy_all:nTF
4321     {
4322         \l_@@_initial_open_bool
4323         \l_@@_final_open_bool
4324         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4325     }
4326     {
4327         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4328         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4329     }

```

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 (?).

```

4330     {
4331         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim

```

```

4332     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4333   }
4334   \@@_draw_line:
4335 }

```

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.

```

4336 \cs_new_protected:Npn \@@_draw_Cdots:nmn #1 #2 #3
4337 {
4338   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4339   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4340   {
4341     \@@_find_extremities_of_line:nmmn { #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.

```

4342     \group_begin:
4343     \@@_open_shorten:
4344     \int_if_zero:nTF { #1 }
4345     { \color { nicematrix-first-row } }
4346     {

```

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.

```

4347         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4348         { \color { nicematrix-last-row } }
4349     }
4350     \keys_set:nn { nicematrix / xdots } { #3 }
4351     \@@_color:o \l_@@_xdots_color_tl
4352     \@@_actually_draw_Cdots:
4353   \group_end:
4354 }
4355 }

```

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`.

```

4356 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4357 {
4358   \bool_if:NTF \l_@@_initial_open_bool
4359   { \@@_open_x_initial_dim: }
4360   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4361   \bool_if:NTF \l_@@_final_open_bool
4362   { \@@_open_x_final_dim: }
4363   { \@@_set_final_coords_from_anchor:n { mid-west } }
4364   \bool_lazy_and:nnTF
4365   \l_@@_initial_open_bool
4366   \l_@@_final_open_bool
4367   {
4368     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4369     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4370     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4371     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }

```

```

4372     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4373   }
4374   {
4375     \bool_if:NT \l_@@_initial_open_bool
4376     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4377     \bool_if:NT \l_@@_final_open_bool
4378     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4379   }
4380   \@@_draw_line:
4381 }

4382 \cs_new_protected:Npn \@@_open_y_initial_dim:
4383 {
4384   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4385   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4386   {
4387     \cs_if_exist:cT
4388     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4389     {
4390       \pgfpointanchor
4391       { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4392       { north }
4393       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4394       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4395     }
4396   }
4397   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4398   {
4399     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4400     \dim_set:Nn \l_@@_y_initial_dim
4401     {
4402       \fp_to_dim:n
4403       {
4404         \pgf@y
4405         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4406       }
4407     }
4408   }
4409 }

4410 \cs_new_protected:Npn \@@_open_y_final_dim:
4411 {
4412   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4413   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4414   {
4415     \cs_if_exist:cT
4416     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4417     {
4418       \pgfpointanchor
4419       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4420       { south }
4421       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4422       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4423     }
4424   }
4425   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4426   {
4427     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4428     \dim_set:Nn \l_@@_y_final_dim
4429     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4430   }
4431 }

```

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.

```

4432 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4433 {
4434   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4435   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4436   {
4437     \@@_find_extremities_of_line:nmmm { #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.

```

4438     \group_begin:
4439     \@@_open_shorten:
4440     \int_if_zero:nTF { #2 }
4441     { \color { nicematrix-first-col } }
4442     {
4443       \int_compare:nNnT { #2 } = \l_@@_last_col_int
4444       { \color { nicematrix-last-col } }
4445     }
4446     \keys_set:nn { nicematrix / xdots } { #3 }
4447     \@@_color:o \l_@@_xdots_color_tl
4448     \@@_actually_draw_Vdots:
4449   \group_end:
4450 }
4451 }

```

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`.

```

4452 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4453 {

```

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

```

4454   \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.

```

4455   {
4456     \@@_open_y_initial_dim:
4457     \@@_open_y_final_dim:
4458     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4459     {
4460       \@@_qpoint:n { col - 1 }
4461       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4462       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4463       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4464       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4465     }
4466     {
4467       \bool_lazy_and:nnTF
4468       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4469       { \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”.

```

4470     {
4471         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4472         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4473         \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4474         \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4475         \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4476     }

```

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

```

4477     {
4478         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4479         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4480         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4481         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4482     }
4483 }
4484 }

```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The boolean `\l_tmpa_bool` will indicate whether the column is of type l or may be considered as if.

```

4485     {
4486         \bool_set_false:N \l_tmpa_bool
4487         \bool_if:NF \l_@@_initial_open_bool
4488         {
4489             \bool_if:NF \l_@@_final_open_bool
4490             {
4491                 \@@_set_initial_coords_from_anchor:n { south-west }
4492                 \@@_set_final_coords_from_anchor:n { north-west }
4493                 \bool_set:Nn \l_tmpa_bool
4494                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4495             }
4496         }

```

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

```

4497     \bool_if:NTF \l_@@_initial_open_bool
4498     {
4499         \@@_open_y_initial_dim:
4500         \@@_set_final_coords_from_anchor:n { north }
4501         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4502     }
4503     {
4504         \@@_set_initial_coords_from_anchor:n { south }
4505         \bool_if:NTF \l_@@_final_open_bool
4506         \@@_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.

```

4507     {
4508         \@@_set_final_coords_from_anchor:n { north }
4509         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4510         {
4511             \dim_set:Nn \l_@@_x_initial_dim
4512             {
4513                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4514                 \l_@@_x_initial_dim \l_@@_x_final_dim
4515             }
4516         }
4517     }
4518 }
4519 }
4520 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4521 \@@_draw_line:
4522 }

```

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.

```

4523 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4524 {
4525   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4526   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4527   {
4528     \@@_find_extremities_of_line:nmmn { #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.

```

4529   \group_begin:
4530   \@@_open_shorten:
4531   \keys_set:nn { nicematrix / xdots } { #3 }
4532   \@@_color:o \l_@@_xdots_color_tl
4533   \@@_actually_draw_Ddots:
4534   \group_end:
4535 }
4536 }

```

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`.

```

4537 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4538 {
4539   \bool_if:NTF \l_@@_initial_open_bool
4540   {
4541     \@@_open_y_initial_dim:
4542     \@@_open_x_initial_dim:
4543   }
4544   { \@@_set_initial_coords_from_anchor:n { south-east } }
4545   \bool_if:NTF \l_@@_final_open_bool
4546   {
4547     \@@_open_x_final_dim:
4548     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4549   }
4550   { \@@_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.

```

4551   \bool_if:NT \l_@@_parallelize_diags_bool
4552   {
4553     \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).

```

4554     \int_compare:nNnTF \g_@@_ddots_int = \c_one_int

```

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

```

4555     {

```

```

4556     \dim_gset:Nn \g_@@_delta_x_one_dim
4557     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4558     \dim_gset:Nn \g_@@_delta_y_one_dim
4559     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4560   }

```

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`.

```

4561     {
4562       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4563       {
4564         \dim_set:Nn \l_@@_y_final_dim
4565         {
4566           \l_@@_y_initial_dim +
4567           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4568           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4569         }
4570       }
4571     }
4572   }
4573   \@@_draw_line:
4574 }

```

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.

```

4575 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4576 {
4577   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4578   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4579   {
4580     \@@_find_extremities_of_line:nmmm { #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.

```

4581     \group_begin:
4582     \@@_open_shorten:
4583     \keys_set:nn { nicematrix / xdots } { #3 }
4584     \@@_color:o \l_@@_xdots_color_tl
4585     \@@_actually_draw_Iddots:
4586   \group_end:
4587 }
4588 }

```

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`.

```

4589 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4590 {
4591   \bool_if:NTF \l_@@_initial_open_bool
4592   {
4593     \@@_open_y_initial_dim:
4594     \@@_open_x_initial_dim:
4595   }

```



```

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

```

17 The actual instructions for drawing the dotted lines with Tikz

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

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

```

4627 \cs_new_protected:Npn \l_@@_draw_line:
4628   {
4629     \pgfrememberpicturepositiononpagetrue
4630     \pgf@relevantforpicturesizefalse
4631     \bool_lazy_or:nnTF
4632     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4633     \l_@@_dotted_bool
4634     \l_@@_draw_standard_dotted_line:
4635     \l_@@_draw_unstandard_dotted_line:
4636   }

```

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.

```

4637 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4638 {
4639   \begin { scope }
4640   \@@_draw_unstandard_dotted_line:o
4641     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4642 }

```

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.

```

4643 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4644 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4645 {
4646   \@@_draw_unstandard_dotted_line:noop
4647     { #1 }
4648   \l_@@_xdots_up_tl
4649   \l_@@_xdots_down_tl
4650   \l_@@_xdots_middle_tl
4651 }

```

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

```

4652 \hook_gput_code:nnn { begindocument } { . }
4653 {
4654   \IfPackageLoadedT { tikz }
4655   {
4656     \tikzset
4657     {
4658       @@_node_above / .style = { sloped , above } ,
4659       @@_node_below / .style = { sloped , below } ,
4660       @@_node_middle / .style =
4661       {
4662         sloped ,
4663         inner~sep = \c_@@_innersep_middle_dim
4664       }
4665     }
4666   }
4667 }

4668 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { n o o }
4669 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1 #2 #3 #4
4670 {

```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` “by hand” because, when we use the key `shorten >` and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

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

```

4671   \dim_zero_new:N \l_@@_l_dim
4672   \dim_set:Nn \l_@@_l_dim
4673   {
4674     \fp_to_dim:n
4675     {
4676       sqrt
4677       (
4678         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4679         +
4680         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4681       )

```

```

4682     }
4683 }

```

It seems that, during the first compilations, the value of `\l_@@_l_dim` may be erroneous (equal to zero or very large). We must detect these cases because they would cause errors during the drawing of the dotted line. Maybe we should also write something in the `aux` file to say that one more compilation should be done.

```

4684 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4685 {
4686   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4687   \@@_draw_unstandard_dotted_line_i:
4688 }

```

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

```

4689 \bool_if:NT \l_@@_xdots_h_labels_bool
4690 {
4691   \tikzset
4692   {
4693     @@_node_above / .style = { auto = left } ,
4694     @@_node_below / .style = { auto = right } ,
4695     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4696   }
4697 }
4698 \tl_if_empty:nF { #4 }
4699 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4700 \draw
4701 [ #1 ]
4702 ( \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).

```

4703   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4704   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4705   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4706   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4707 \end { scope }
4708 }
4709 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4710 {
4711   \dim_set:Nn \l_tmpa_dim
4712   {
4713     \l_@@_x_initial_dim
4714     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4715     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4716   }
4717   \dim_set:Nn \l_tmpb_dim
4718   {
4719     \l_@@_y_initial_dim
4720     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4721     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4722   }
4723   \dim_set:Nn \l_@@_tmpc_dim
4724   {
4725     \l_@@_x_final_dim
4726     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4727     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4728   }
4729   \dim_set:Nn \l_@@_tmpd_dim
4730   {
4731     \l_@@_y_final_dim
4732     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4733     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4734   }
4735   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim

```

```

4736 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4737 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4738 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4739 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4740 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4741 {
4742   \group_begin:

```

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

```

4743   \dim_zero_new:N \l_@@_l_dim
4744   \dim_set:Nn \l_@@_l_dim
4745   {
4746     \fp_to_dim:n
4747     {
4748       sqrt
4749       (
4750         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4751         +
4752         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4753       )
4754     }
4755   }

```

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.

```

4756   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4757   {
4758     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4759     \@@_draw_standard_dotted_line_i:
4760   }
4761   \group_end:
4762   \bool_lazy_all:nF
4763   {
4764     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4765     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4766     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4767   }
4768   \l_@@_labels_standard_dotted_line:
4769 }
4770 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4771 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4772 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4773   \int_set:Nn \l_tmpa_int
4774   {
4775     \dim_ratio:nn
4776     {
4777       \l_@@_l_dim
4778       - \l_@@_xdots_shorten_start_dim
4779       - \l_@@_xdots_shorten_end_dim
4780     }
4781     \l_@@_xdots_inter_dim
4782   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4783   \dim_set:Nn \l_tmpa_dim
4784   {
4785     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4786     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4787   }
4788   \dim_set:Nn \l_tmpb_dim
4789   {
4790     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4791     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4792   }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4793   \dim_gadd:Nn \l_@@_x_initial_dim
4794   {
4795     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4796     \dim_ratio:nn
4797     {
4798       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4799       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4800     }
4801     { 2 \l_@@_l_dim }
4802   }
4803   \dim_gadd:Nn \l_@@_y_initial_dim
4804   {
4805     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4806     \dim_ratio:nn
4807     {
4808       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4809       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4810     }
4811     { 2 \l_@@_l_dim }
4812   }
4813   \pgf@relevantforpicturesizefalse
4814   \int_step_inline:nnn \c_zero_int \l_tmpa_int
4815   {
4816     \pgfpathcircle
4817     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4818     { \l_@@_xdots_radius_dim }
4819     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4820     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4821   }
4822   \pgfusepathqfill
4823 }

```

```

4824 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4825 {
4826   \pgfscope
4827   \pgftransformshift
4828   {
4829     \pgfpointlineattime { 0.5 }
4830     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4831     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4832   }
4833   \fp_set:Nn \l_tmpa_fp
4834   {
4835     atand
4836     (
4837       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4838       \l_@@_x_final_dim - \l_@@_x_initial_dim
4839     )

```

```

4840 }
4841 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4842 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4843 \tl_if_empty:NF \l_@@_xdots_middle_tl
4844 {
4845   \begin { pgfscope }
4846   \pgfset { inner-sep = \c_@@_innersep_middle_dim }
4847   \pgfnode
4848   { rectangle }
4849   { center }
4850   {
4851     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4852     {
4853       \c_math_toggle_token
4854       \scriptstyle \l_@@_xdots_middle_tl
4855       \c_math_toggle_token
4856     }
4857   }
4858   { }
4859   {
4860     \pgfsetfillcolor { white }
4861     \pgfusepath { fill }
4862   }
4863   \end { pgfscope }
4864 }
4865 \tl_if_empty:NF \l_@@_xdots_up_tl
4866 {
4867   \pgfnode
4868   { rectangle }
4869   { south }
4870   {
4871     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4872     {
4873       \c_math_toggle_token
4874       \scriptstyle \l_@@_xdots_up_tl
4875       \c_math_toggle_token
4876     }
4877   }
4878   { }
4879   { \pgfusepath { } }
4880 }
4881 \tl_if_empty:NF \l_@@_xdots_down_tl
4882 {
4883   \pgfnode
4884   { rectangle }
4885   { north }
4886   {
4887     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4888     {
4889       \c_math_toggle_token
4890       \scriptstyle \l_@@_xdots_down_tl
4891       \c_math_toggle_token
4892     }
4893   }
4894   { }
4895   { \pgfusepath { } }
4896 }
4897 \endpgfscope
4898 }

```

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.

```

4899 \hook_gput_code:nnn { begindocument } { . }
4900 {
4901   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { } { } { } }
4902   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4903   \cs_new_protected:Npn \@@_Ldots
4904     { \@@_collect_options:n { \@@_Ldots_i } }
4905   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4906     {
4907       \int_if_zero:nTF \c@jCol
4908         { \@@_error:nn { in~first~col } \Ldots }
4909         {
4910           \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4911             { \@@_error:nn { in~last~col } \Ldots }
4912             {
4913               \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4914               { #1 , down = #2 , up = #3 , middle = #4 }
4915             }
4916         }
4917       \bool_if:NF \l_@@_nullify_dots_bool
4918         { \phantom { \ensuremath { \@@_old_ldots } } }
4919       \bool_gset_true:N \g_@@_empty_cell_bool
4920     }

4921   \cs_new_protected:Npn \@@_Cdots
4922     { \@@_collect_options:n { \@@_Cdots_i } }
4923   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4924     {
4925       \int_if_zero:nTF \c@jCol
4926         { \@@_error:nn { in~first~col } \Cdots }
4927         {
4928           \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4929             { \@@_error:nn { in~last~col } \Cdots }
4930             {
4931               \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4932               { #1 , down = #2 , up = #3 , middle = #4 }
4933             }
4934         }
4935       \bool_if:NF \l_@@_nullify_dots_bool
4936         { \phantom { \ensuremath { \@@_old_cdots } } }
4937       \bool_gset_true:N \g_@@_empty_cell_bool
4938     }

4939   \cs_new_protected:Npn \@@_Vdots
4940     { \@@_collect_options:n { \@@_Vdots_i } }
4941   \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4942     {
4943       \int_if_zero:nTF \c@iRow
4944         { \@@_error:nn { in~first~row } \Vdots }
4945         {

```

```

4946     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4947     { \@@_error:nn { in~last~row } \Vdots }
4948     {
4949         \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4950         { #1 , down = #2 , up = #3 , middle = #4 }
4951     }
4952 }
4953 \bool_if:NF \l_@@_nullify_dots_bool
4954 { \phantom { \ensuremath { \@@_old_vdots } } }
4955 \bool_gset_true:N \g_@@_empty_cell_bool
4956 }

4957 \cs_new_protected:Npn \@@_Ddots
4958 { \@@_collect_options:n { \@@_Ddots_i } }
4959 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4960 {
4961     \int_case:nnF \c@iRow
4962     {
4963         0 { \@@_error:nn { in~first~row } \Ddots }
4964         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4965     }
4966     {
4967         \int_case:nnF \c@jCol
4968         {
4969             0 { \@@_error:nn { in~first~col } \Ddots }
4970             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4971         }
4972         {
4973             \keys_set_known:nn { nicematrix / Ddots } { #1 }
4974             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4975             { #1 , down = #2 , up = #3 , middle = #4 }
4976         }
4977     }
4978 }
4979 \bool_if:NF \l_@@_nullify_dots_bool
4980 { \phantom { \ensuremath { \@@_old_ddots } } }
4981 \bool_gset_true:N \g_@@_empty_cell_bool
4982 }

4983 \cs_new_protected:Npn \@@_Iddots
4984 { \@@_collect_options:n { \@@_Iddots_i } }
4985 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4986 {
4987     \int_case:nnF \c@iRow
4988     {
4989         0 { \@@_error:nn { in~first~row } \Iddots }
4990         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4991     }
4992     {
4993         \int_case:nnF \c@jCol
4994         {
4995             0 { \@@_error:nn { in~first~col } \Iddots }
4996             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4997         }
4998         {
4999             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5000             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5001             { #1 , down = #2 , up = #3 , middle = #4 }
5002         }
5003     }
5004 }
5005 \bool_if:NF \l_@@_nullify_dots_bool
5006 { \phantom { \ensuremath { \@@_old_iddots } } }

```



```

5006     \bool_gset_true:N \g_@@_empty_cell_bool
5007   }
5008 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5009 \keys_define:nn { nicematrix / Ddots }
5010 {
5011   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5012   draw-first .default:n = true ,
5013   draw-first .value_forbidden:n = true
5014 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5015 \cs_new_protected:Npn \@@_Hspace:
5016 {
5017   \bool_gset_true:N \g_@@_empty_cell_bool
5018   \hspace
5019 }

```

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.

```

5020 \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.

```

5021 \cs_new:Npn \@@_Hdotsfor:
5022 {
5023   \bool_lazy_and:nnTF
5024     { \int_if_zero_p:n \c@jCol }
5025     { \int_if_zero_p:n \l_@@_first_col_int }
5026     {
5027       \bool_if:NTF \g_@@_after_col_zero_bool
5028       {
5029         \multicolumn { 1 } { c } { }
5030         \@@_Hdotsfor_i
5031       }
5032       { \@@_fatal:n { Hdotsfor~in~col-0 } }
5033     }
5034     {
5035       \multicolumn { 1 } { c } { }
5036       \@@_Hdotsfor_i
5037     }
5038 }

```

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:).

```

5039 \hook_gput_code:nnn { begindocument } { . }
5040 {
5041   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5042   \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.

```

5043   \cs_new_protected:Npn \@@_Hdotsfor_i
5044     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5045   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5046     {

```

```

5047 \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5048 {
5049   \@@_Hdotsfor:nmmn
5050   { \int_use:N \c@iRow }
5051   { \int_use:N \c@jCol }
5052   { #2 }
5053   {
5054     #1 , #3 ,
5055     down = \exp_not:n { #4 } ,
5056     up = \exp_not:n { #5 } ,
5057     middle = \exp_not:n { #6 }
5058   }
5059 }
5060 \prg_replicate:nn { #2 - 1 }
5061 {
5062   &
5063   \multicolumn { 1 } { c } { }
5064   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5065 }
5066 }
5067 }

```

```

5068 \cs_new_protected:Npn \@@_Hdotsfor:nmmn #1 #2 #3 #4
5069 {
5070   \bool_set_false:N \l_@@_initial_open_bool
5071   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5072 \int_set:Nn \l_@@_initial_i_int { #1 }
5073 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5074 \int_compare:nNnTF { #2 } = \c_one_int
5075 {
5076   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5077   \bool_set_true:N \l_@@_initial_open_bool
5078 }
5079 {
5080   \cs_if_exist:cTF
5081   {
5082     pgf @ sh @ ns @ \@@_env:
5083     - \int_use:N \l_@@_initial_i_int
5084     - \int_eval:n { #2 - 1 }
5085   }
5086   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5087   {
5088     \int_set:Nn \l_@@_initial_j_int { #2 }
5089     \bool_set_true:N \l_@@_initial_open_bool
5090   }
5091 }
5092 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5093 {
5094   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5095   \bool_set_true:N \l_@@_final_open_bool
5096 }
5097 {
5098   \cs_if_exist:cTF
5099   {
5100     pgf @ sh @ ns @ \@@_env:
5101     - \int_use:N \l_@@_final_i_int
5102     - \int_eval:n { #2 + #3 }
5103   }
5104   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5105   {

```

```

5106         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5107         \bool_set_true:N \l_@@_final_open_bool
5108     }
5109 }

5110 \group_begin:
5111 \@@_open_shorten:
5112 \int_if_zero:nTF { #1 }
5113   { \color { nicematrix-first-row } }
5114   {
5115     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5116     { \color { nicematrix-last-row } }
5117   }
5118
5119 \keys_set:nn { nicematrix / xdots } { #4 }
5120 \@@_color:o \l_@@_xdots_color_tl
5121 \@@_actually_draw_Ldots:
5122 \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).

```

5123     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5124     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5125 }

5126 \hook_gput_code:nnn { begindocument } { . }
5127 {
5128   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5129   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5130   \cs_new_protected:Npn \@@_Vdotsfor:
5131     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5132   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5133     {
5134       \bool_gset_true:N \g_@@_empty_cell_bool
5135       \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5136         {
5137           \@@_Vdotsfor:nnnn
5138             { \int_use:N \c@iRow }
5139             { \int_use:N \c@jCol }
5140             { #2 }
5141             {
5142               #1 , #3 ,
5143               down = \exp_not:n { #4 } ,
5144               up = \exp_not:n { #5 } ,
5145               middle = \exp_not:n { #6 }
5146             }
5147         }
5148     }
5149 }

5150 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5151 {
5152   \bool_set_false:N \l_@@_initial_open_bool
5153   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5154     \int_set:Nn \l_@@_initial_j_int { #2 }
5155     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5156   \int_compare:nNnTF { #1 } = \c_one_int
5157   {
5158     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5159     \bool_set_true:N \l_@@_initial_open_bool
5160   }
5161   {
5162     \cs_if_exist:cTF
5163     {
5164       pgf @ sh @ ns @ \@@_env:
5165       - \int_eval:n { #1 - 1 }
5166       - \int_use:N \l_@@_initial_j_int
5167     }
5168     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5169     {
5170       \int_set:Nn \l_@@_initial_i_int { #1 }
5171       \bool_set_true:N \l_@@_initial_open_bool
5172     }
5173   }
5174   \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5175   {
5176     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5177     \bool_set_true:N \l_@@_final_open_bool
5178   }
5179   {
5180     \cs_if_exist:cTF
5181     {
5182       pgf @ sh @ ns @ \@@_env:
5183       - \int_eval:n { #1 + #3 }
5184       - \int_use:N \l_@@_final_j_int
5185     }
5186     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5187     {
5188       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5189       \bool_set_true:N \l_@@_final_open_bool
5190     }
5191   }
5192   \group_begin:
5193   \@@_open_shorten:
5194   \int_if_zero:nTF { #2 }
5195   { \color { nicematrix-first-col } }
5196   {
5197     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5198     { \color { nicematrix-last-col } }
5199   }
5200   \keys_set:nn { nicematrix / xdots } { #4 }
5201   \@@_color:o \l_@@_xdots_color_tl
5202   \@@_actually_draw_Vdots:
5203   \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5204   \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5205   { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5206 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5207 \NewDocumentCommand \@@_rotate: { 0 { } }
5208 {

```

```

5209 \peek_remove_spaces:n
5210 {
5211   \bool_gset_true:N \g_@@_rotate_bool
5212   \keys_set:nn { nicematrix / rotate } { #1 }
5213 }
5214 }

5215 \keys_define:nn { nicematrix / rotate }
5216 {
5217   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5218   c .value_forbidden:n = true ,
5219   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5220 }

```

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).¹³

```

5221 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5222 {
5223   \tl_if_empty:nTF { #2 }
5224   { #1 }
5225   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5226 }
5227 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5228 { \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).

```

5229 \hook_gput_code:nnn { begindocument } { . }
5230 {
5231   \cs_set_nopar:Npn \l_@@_argspec_tl
5232   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5233   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5234   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5235   {
5236     \group_begin:
5237     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5238     \@@_color:o \l_@@_xdots_color_tl
5239     \use:e
5240     {
5241       \@@_line_i:nn
5242       { \@@_double_int_eval:n #2 - \q_stop }
5243       { \@@_double_int_eval:n #3 - \q_stop }

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5244     }
5245     \group_end:
5246   }
5247 }

5248 \cs_new_protected:Npn \@@_line_ii:nn #1 #2
5249 {
5250   \bool_set_false:N \l_@@_initial_open_bool
5251   \bool_set_false:N \l_@@_final_open_bool
5252   \bool_lazy_or:nnTF
5253     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5254     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5255     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5256     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5257   }

5258 \hook_gput_code:nnn { begindocument } { . }
5259 {
5260   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5261   {

```

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:`.

```

5262     \c_@@_pgfortikzpicture_tl
5263     \@@_draw_line_iii:nn { #1 } { #2 }
5264     \c_@@_endpgfortikzpicture_tl
5265   }
5266 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5267 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5268 {
5269   \pgfrememberpicturepositiononpagetrue
5270   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5271   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5272   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5273   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5274   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5275   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5276   \@@_draw_line:
5277 }

```

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

```
5278 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5279 { \int_compare:nNt { \c@iRow } < { #1 } { #2 } }
```

\@@_put_in_row_style will be used several times by \RowStyle.

```
5280 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5281 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5282 {
5283   \tl_gput_right:Ne \g_@@_row_style_tl
5284   {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```
5285     \exp_not:N
5286     \@@_if_row_less_than:nn
5287     { \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).

```
5288     { \exp_not:n { #1 } \scan_stop: }
5289   }
5290 }
```

```
5291 \keys_define:nn { nicematrix / RowStyle }
5292 {
5293   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5294   cell-space-top-limit .value_required:n = true ,
5295   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5296   cell-space-bottom-limit .value_required:n = true ,
5297   cell-space-limits .meta:n =
5298   {
5299     cell-space-top-limit = #1 ,
5300     cell-space-bottom-limit = #1 ,
5301   } ,
5302   color .tl_set:N = \l_@@_color_tl ,
5303   color .value_required:n = true ,
5304   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5305   bold .default:n = true ,
5306   nb-rows .code:n =
5307     \str_if_eq:eeTF { #1 } { * }
5308     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5309     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5310   nb-rows .value_required:n = true ,
5311   rowcolor .tl_set:N = \l_tmpa_tl ,
5312   rowcolor .value_required:n = true ,
5313   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5314 }
```

```
5315 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5316 {
5317   \group_begin:
5318   \tl_clear:N \l_tmpa_tl
5319   \tl_clear:N \l_@@_color_tl
5320   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5321   \dim_zero:N \l_tmpa_dim
5322   \dim_zero:N \l_tmpb_dim
5323   \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key rowcolor has been used.

```
5324   \tl_if_empty:NF \l_tmpa_tl
5325   {
```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```
5326     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5327     {
```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5328         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5329         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5330         { \int_use:N \c@iRow - * }
5331     }
```

Then, the other rows (if there is several rows).

```
5332     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5333     {
5334         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5335         {
5336             \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5337             {
5338                 \int_eval:n { \c@iRow + 1 }
5339                 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5340             }
5341         }
5342     }
5343 }
5344 \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```
5345     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5346     {
5347         \@@_put_in_row_style:e
5348         {
5349             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5350             {
```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```
5351                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5352                 { \dim_use:N \l_tmpa_dim }
5353             }
5354         }
5355     }
```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```
5356     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5357     {
5358         \@@_put_in_row_style:e
5359         {
5360             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5361             {
5362                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5363                 { \dim_use:N \l_tmpb_dim }
5364             }
5365         }
5366     }
```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```
5367     \tl_if_empty:NF \l_@@_color_tl
5368     {
5369         \@@_put_in_row_style:e
5370         {
5371             \mode_leave_vertical:
5372             \@@_color:n { \l_@@_color_tl }
5373         }
5374     }
```


`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5375   \bool_if:NT \l_@@_bold_row_style_bool
5376   {
5377     \@@_put_in_row_style:n
5378     {
5379       \exp_not:n
5380       {
5381         \if_mode_math:
5382         \c_math_toggle_token
5383         \bfseries \boldmath
5384         \c_math_toggle_token
5385         \else:
5386         \bfseries \boldmath
5387         \fi:
5388       }
5389     }
5390   }
5391   \group_end:
5392   \g_@@_row_style_tl
5393   \ignorespaces
5394 }

```

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).

```

5395 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5396 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5397 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5398 {

```

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.

```

5399   \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`.

```

5400   \str_if_in:nnF { #1 } { !! }
5401   {
5402     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5403     { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5404     }
5405     \int_if_zero:nTF \l_tmpa_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5406     {
5407     \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5408     \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5409     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5410     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5411     }

```

The following command must be used within a `\pgfpicture`.

```

5412 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5413 {
5414     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5415     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5416     \group_begin:
5417     \pgfsetcornersarced
5418     {
5419     \pgfpoint
5420     { \l_@@_tab_rounded_corners_dim }
5421     { \l_@@_tab_rounded_corners_dim }
5422     }

```

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.

```

5423     \bool_if:NTF \l_@@_hvlines_bool
5424     {
5425     \pgfpathrectanglecorners
5426     {
5427     \pgfpointadd
5428     { \@@_qpoint:n { row-1 } }
5429     { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5430     }
5431     {
5432     \pgfpointadd
5433     {
5434     \@@_qpoint:n
5435     { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5436     }
5437     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5438     }
5439     }
5440     {
5441     \pgfpathrectanglecorners
5442     { \@@_qpoint:n { row-1 } }
5443     {
5444     \pgfpointadd
5445     {
5446     \@@_qpoint:n
5447     { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5448     }
5449     { \pgfpoint \c_zero_dim \arrayrulewidth }
5450     }

```

```

5451     }
5452     \pgfusepath { clip }
5453     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5454     }
5455 }

```

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`).

```

5456 \cs_new_protected:Npn \@@_actually_color:
5457 {
5458   \pgfpicture
5459   \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.

```

5460     \@@_clip_with_rounded_corners:
5461     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5462     {
5463       \int_compare:nNnTF { ##1 } = \c_one_int
5464       {
5465         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5466         \use:c { g_@@_color _ 1 _tl }
5467         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5468       }
5469       {
5470         \begin { pgfscope }
5471           \@@_color_opacity ##2
5472           \use:c { g_@@_color _ ##1 _tl }
5473           \tl_gclear:c { g_@@_color _ ##1 _tl }
5474           \pgfusepath { fill }
5475           \end { pgfscope }
5476       }
5477     }
5478   \endpgfpicture
5479 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5480 \cs_new_protected:Npn \@@_color_opacity
5481 {
5482   \peek_meaning:NTF [
5483     { \@@_color_opacity:w }
5484     { \@@_color_opacity:w [ ] }
5485   }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5486 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5487 {
5488   \tl_clear:N \l_tmpa_tl
5489   \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.

```

5490   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5491   \tl_if_empty:NTF \l_tmpb_tl
5492     { \@declaredcolor }
5493     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5494 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5495 \keys_define:nn { nicematrix / color-opacity }
5496 {
5497   opacity .tl_set:N          = \l_tmpa_tl ,
5498   opacity .value_required:n = true
5499 }

5500 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5501 {
5502   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5503   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5504   \@@_cartesian_path:
5505 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5506 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5507 {
5508   \tl_if_blank:nF { #2 }
5509   {
5510     \@@_add_to_colors_seq:en
5511     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5512     { \@@_cartesian_color:nn { #3 } { - } }
5513   }
5514 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5515 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5516 {
5517   \tl_if_blank:nF { #2 }
5518   {
5519     \@@_add_to_colors_seq:en
5520     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5521     { \@@_cartesian_color:nn { - } { #3 } }
5522   }
5523 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5524 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5525 {
5526   \tl_if_blank:nF { #2 }
5527   {
5528     \@@_add_to_colors_seq:en
5529     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5530     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5531   }
5532 }

```

The last argument is the radius of the corners of the rectangle.

```

5533 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5534 {
5535   \tl_if_blank:nF { #2 }
5536   {
5537     \@@_add_to_colors_seq:en
5538     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5539     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5540   }
5541 }

```

The last argument is the radius of the corners of the rectangle.

```

5542 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5543 {
5544   \@@_cut_on_hyphen:w #1 \q_stop
5545   \tl_clear_new:N \l_@@_tmpc_tl
5546   \tl_clear_new:N \l_@@_tmpd_tl
5547   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5548   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5549   \@@_cut_on_hyphen:w #2 \q_stop
5550   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5551   \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`.

```

5552   \@@_cartesian_path:n { #3 }
5553 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5554 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5555 {
5556   \clist_map_inline:nn { #3 }
5557   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5558 }

```

```

5559 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5560 {
5561   \int_step_inline:nn \c@iRow
5562   {
5563     \int_step_inline:nn \c@jCol
5564     {
5565       \int_if_even:nTF { #####1 + ##1 }
5566       { \@@_cellcolor [ #1 ] { #2 } }
5567       { \@@_cellcolor [ #1 ] { #3 } }
5568       { ##1 - #####1 }
5569     }
5570   }
5571 }

```

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”.

```

5572 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5573 {
5574   \@@_rectanglecolor [ #1 ] { #2 }
5575   { 1 - 1 }
5576   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5577 }

5578 \keys_define:nn { nicematrix / rowcolors }
5579 {
5580   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5581   respect-blocks .default:n = true ,
5582   cols .tl_set:N = \l_@@_cols_tl ,
5583   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5584   restart .default:n = true ,
5585   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5586 }

```

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*.

```
5587 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5588 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
5589 \group_begin:
5590 \seq_clear_new:N \l_@@_colors_seq
5591 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5592 \tl_clear_new:N \l_@@_cols_tl
5593 \cs_set_nopar:Npn \l_@@_cols_tl { - }
5594 \keys_set:mn { 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).

```
5595 \int_zero_new:N \l_@@_color_int
5596 \int_set_eq:NN \l_@@_color_int \c_one_int
5597 \bool_if:NT \l_@@_respect_blocks_bool
5598 {
```

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`).

```
5599 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5600 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5601 { \@@_not_in_exterior_p:nnnn ##1 }
5602 }
5603 \pgfpicture
5604 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5605 \clist_map_inline:nn { #2 }
5606 {
5607 \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5608 \tl_if_in:NnTF \l_tmpa_tl { - }
5609 { \@@_cut_on_hyphen:w ##1 \q_stop }
5610 { \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.

```
5611 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5612 \int_set:Nn \l_@@_color_int
5613 { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5614 \int_zero_new:N \l_@@_tmpc_int
5615 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5616 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5617 {
```

We will compute in `\l_tmpb_int` the last row of the "block".

```
5618 \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).

```
5619 \bool_if:NT \l_@@_respect_blocks_bool
5620 {
5621 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5622 { \@@_intersect_our_row_p:nnnn ###1 }
5623 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn ###1 }
```

Now, the last row of the block is computed in `\l_tmpb_int`.

```
5624 }
5625 \tl_set:No \l_@@_rows_tl
5626 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5627         \tl_clear_new:N \l_@@_color_tl
5628         \tl_set:Ne \l_@@_color_tl
5629         {
5630             \@@_color_index:n
5631             {
5632                 \int_mod:nn
5633                 { \l_@@_color_int - 1 }
5634                 { \seq_count:N \l_@@_colors_seq }
5635                 + 1
5636             }
5637         }
5638         \tl_if_empty:NF \l_@@_color_tl
5639         {
5640             \@@_add_to_colors_seq:ee
5641             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5642             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5643         }
5644         \int_incr:N \l_@@_color_int
5645         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5646     }
5647 }
5648 \endpgfpicture
5649 \group_end:
5650 }

```

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.

```

5651 \cs_new:Npn \@@_color_index:n #1
5652 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5653     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5654     { \@@_color_index:n { #1 - 1 } }
5655     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5656 }

```

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.

```

5657 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5658 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5659 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5660 {
5661     \int_compare:nNnT { #3 } > \l_tmpb_int
5662     { \int_set:Nn \l_tmpb_int { #3 } }
5663 }

```

```

5664 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5665 {
5666     \int_if_zero:nTF { #4 }
5667     \prg_return_false:
5668     {
5669         \int_compare:nNnTF { #2 } > \c@jCol
5670         \prg_return_false:
5671         \prg_return_true:
5672     }
5673 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5674 \prg_new_conditional:Nnn \@_intersect_our_row:nnnnn p
5675 {
5676   \int_compare:nNnTF { #1 } > \l_tmpa_int
5677     \prg_return_false:
5678     {
5679       \int_compare:nNnTF \l_tmpa_int > { #3 }
5680         \prg_return_false:
5681         \prg_return_true:
5682     }
5683 }

```

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).

```

5684 \cs_new_protected:Npn \@_cartesian_path_normal:n #1
5685 {
5686   \dim_compare:nNnTF { #1 } = \c_zero_dim
5687     {
5688       \bool_if:NTF
5689         \l_@@_nocolor_used_bool
5690         \@_cartesian_path_normal_ii:
5691         {
5692           \clist_if_empty:NTF \l_@@_corners_cells_clist
5693             { \@_cartesian_path_normal_i:n { #1 } }
5694             \@_cartesian_path_normal_ii:
5695         }
5696     }
5697     { \@_cartesian_path_normal_i:n { #1 } }
5698 }

```

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.

```

5699 \cs_new_protected:Npn \@_cartesian_path_normal_i:n #1
5700 {
5701   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5702   \clist_map_inline:Nn \l_@@_cols_tl
5703   {
5704     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5705     \tl_if_in:NnTF \l_tmpa_tl { - }
5706       { \@_cut_on_hyphen:w ##1 \q_stop }
5707       { \@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5708     \tl_if_empty:NTF \l_tmpa_tl
5709       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5710       {
5711         \str_if_eq:eeT \l_tmpa_tl { * }
5712         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5713       }
5714     \tl_if_empty:NTF \l_tmpb_tl
5715       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5716       {
5717         \str_if_eq:eeT \l_tmpb_tl { * }
5718         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5719       }
5720     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5721     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```


`\l_@@_tmpc_tl` will contain the number of column.

```

5722 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5723 \@@_qpoint:n { col - \l_tmpa_tl }
5724 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5725   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5726   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5727 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5728 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5729 \clist_map_inline:Nn \l_@@_rows_tl
5730 {
5731   \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5732   \tl_if_in:NnTF \l_tmpa_tl { - }
5733     { \@@_cut_on_hyphen:w ####1 \q_stop }
5734     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5735   \tl_if_empty:NTF \l_tmpa_tl
5736     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5737     {
5738       \str_if_eq:eeT \l_tmpa_tl { * }
5739       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5740     }
5741   \tl_if_empty:NTF \l_tmpb_tl
5742     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5743     {
5744       \str_if_eq:eeT \l_tmpb_tl { * }
5745       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5746     }
5747   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5748     { \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`.

```

5749 \cs_if_exist:cF
5750 { @@_nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5751 {
5752   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5753   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5754   \@@_qpoint:n { row - \l_tmpa_tl }
5755   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5756   \pgfpathrectanglecorners
5757     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5758     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5759 }
5760 }
5761 }
5762 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5763 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5764 {
5765   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5766   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5767 \clist_map_inline:Nn \l_@@_cols_tl
5768 {
5769   \@@_qpoint:n { col - ##1 }
5770   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5771     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5772     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5773   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5774   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5775     \clist_map_inline:Nn \l_@@_rows_tl
5776     {
5777         \@@_if_in_corner:nF { #####1 - #1 }
5778         {
5779             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5780             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5781             \@@_qpoint:n { row - #####1 }
5782             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5783             \cs_if_exist:cF { @@ _ nocolor _ #####1 - #1 }
5784             {
5785                 \pgfpathrectanglecorners
5786                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5787                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5788             }
5789         }
5790     }
5791 }
5792 }

```

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`).

```

5793 \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

```

5794 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5795 {
5796     \bool_set_true:N \l_@@_nocolor_used_bool
5797     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5798     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5799     \clist_map_inline:Nn \l_@@_rows_tl
5800     {
5801         \clist_map_inline:Nn \l_@@_cols_tl
5802         { \cs_set_nopar:cpn { @@ _ nocolor _ #1 - #####1 } { } }
5803     }
5804 }

```

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.

```

5805 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5806 {
5807     \clist_set_eq:NN \l_tmpa_clist #1
5808     \clist_clear:N #1
5809     \clist_map_inline:Nn \l_tmpa_clist
5810     {
5811         \cs_set_nopar:Npn \l_tmpa_tl { #1 }
5812         \tl_if_in:NnTF \l_tmpa_tl { - }
5813         { \@@_cut_on_hyphen:w #1 \q_stop }
5814         { \@@_cut_on_hyphen:w #1 - #1 \q_stop }
5815         \bool_lazy_or:nnT
5816         { \str_if_eq_p:ee \l_tmpa_tl { * } }
5817         { \tl_if_blank_p:o \l_tmpa_tl }
5818         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5819         \bool_lazy_or:nnT
5820         { \str_if_eq_p:ee \l_tmpb_tl { * } }
5821         { \tl_if_blank_p:o \l_tmpb_tl }

```

```

5822     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5823     \int_compare:nNnT \l_tmpb_tl > #2
5824     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5825     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5826     { \clist_put_right:Nn #1 { #####1 } }
5827   }
5828 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5829 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5830 {
5831   \@@_test_color_inside:
5832   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5833   {

```

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`).

```

5834     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5835     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5836   }
5837   \ignorespaces
5838 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5839 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5840 {
5841   \@@_test_color_inside:
5842   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5843   {
5844     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5845     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5846     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5847   }
5848   \ignorespaces
5849 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5850 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5851 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5852 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5853 {
5854   \@@_test_color_inside:
5855   \peek_remove_spaces:n
5856   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5857 }

5858 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5859 {

```

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`.

```

5860     \seq_gclear:N \g_tmpa_seq
5861     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5862       { \@@_rowlistcolors_tabular_i:nmmn ##1 }
5863     \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`.

```

5864     \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5865     {
5866       { \int_use:N \c@iRow }
5867       { \exp_not:n { #1 } }
5868       { \exp_not:n { #2 } }
5869       { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5870     }
5871   }

```

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`).

```

5872 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nmmn #1 #2 #3 #4
5873 {
5874   \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).

```

5875   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5876   {
5877     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5878     {
5879       \@@_rowlistcolors
5880       [ \exp_not:n { #2 } ]
5881       { #1 - \int_eval:n { \c@iRow - 1 } }
5882       { \exp_not:n { #3 } }
5883       [ \exp_not:n { #4 } ]
5884     }
5885   }
5886 }

```

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.

```

5887 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5888 {
5889   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5890     { \@@_rowlistcolors_tabular_ii:nmmn ##1 }
5891   \seq_gclear:N \g_@@_rowlistcolors_seq
5892 }

```

```

5893 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nmnn #1 #2 #3 #4
5894 {
5895   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5896   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5897 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is written 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.

```

5898 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5899 {

```

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).

```

5900   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5901   {

```

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).

```

5902     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5903     {
5904       \exp_not:N \columncolor [ #1 ]
5905       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5906     }
5907   }
5908 }

```

```

5909 \hook_gput_code:nnn { begindocument } { . }
5910 {
5911   \IfPackageLoadedTF { colortbl }
5912   {
5913     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5914     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5915     \cs_new_protected:Npn \@@_revert_colortbl:
5916     {
5917       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5918       {
5919         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5920         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5921       }
5922     }
5923   }
5924   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5925 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of array) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
5926 \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`.

```
5927 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5928 {
5929   \int_if_zero:nTF \l_@@_first_col_int
5930     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5931     {
5932       \int_if_zero:nTF \c@jCol
5933         {
5934           \int_compare:nNnF \c@iRow = { -1 }
5935             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5936         }
5937         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5938     }
5939 }
```

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.

```
5940 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5941 {
5942   \int_if_zero:nF \c@iRow
5943   {
5944     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5945     {
5946       \int_compare:nNnT \c@jCol > \c_zero_int
5947       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5948     }
5949   }
5950 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5951 \keys_define:nn { nicematrix / Rules }
5952 {
5953   position .int_set:N = \l_@@_position_int ,
5954   position .value_required:n = true ,
5955   start .int_set:N = \l_@@_start_int ,
5956   end .code:n =
5957     \bool_lazy_or:nnTF
5958     { \tl_if_empty_p:n { #1 } }
5959     { \str_if_eq_p:ee { #1 } { last } }
5960     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5961     { \int_set:Nn \l_@@_end_int { #1 } }
5962 }
```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous

rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```

5963 \keys_define:nn { nicematrix / RulesBis }
5964 {
5965   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5966   multiplicity .initial:n = 1 ,
5967   dotted .bool_set:N = \l_@@_dotted_bool ,
5968   dotted .initial:n = false ,
5969   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]`.

```

5970   color .code:n =
5971     \@@_set_CT@arc@:n { #1 }
5972     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5973   color .value_required:n = true ,
5974   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5975   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.

```

5976   tikz .code:n =
5977     \IfPackageLoadedTF { tikz }
5978       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5979       { \@@_error:n { tikz-without-tikz } } ,
5980   tikz .value_required:n = true ,
5981   total-width .dim_set:N = \l_@@_rule_width_dim ,
5982   total-width .value_required:n = true ,
5983   width .meta:n = { total-width = #1 } ,
5984   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5985 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5986 \cs_new_protected:Npn \@@_vline:n #1
5987 {

```

The group is for the options.

```

5988   \group_begin:
5989   \int_set_eq:NN \l_@@_end_int \c@iRow
5990   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5991   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5992     \@@_vline_i:
5993   \group_end:
5994 }
5995 \cs_new_protected:Npn \@@_vline_i:
5996 {

```

`\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`.

```

5997   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
5998   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5999     \l_tmpa_tl
6000   {

```

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.

```

6001     \bool_gset_true:N \g_tmpa_bool
6002     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6003     { \@@_test_vline_in_block:nnnnn ##1 }
6004     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6005     { \@@_test_vline_in_block:nnnnn ##1 }
6006     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6007     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6008     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6009     \bool_if:NTF \g_tmpa_bool
6010     {
6011         \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.

```

6012         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6013     }
6014     {
6015         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6016         {
6017             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6018             \@@_vline_ii:
6019             \int_zero:N \l_@@_local_start_int
6020         }
6021     }
6022 }
6023 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6024 {
6025     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6026     \@@_vline_ii:
6027 }
6028 }

```

```

6029 \cs_new_protected:Npn \@@_test_in_corner_v:
6030 {
6031     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6032     {
6033         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6034         { \bool_set_false:N \g_tmpa_bool }
6035     }
6036     {
6037         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6038         {
6039             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6040             { \bool_set_false:N \g_tmpa_bool }
6041             {
6042                 \@@_if_in_corner:nT
6043                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6044                 { \bool_set_false:N \g_tmpa_bool }
6045             }
6046         }
6047     }
6048 }

```

```

6049 \cs_new_protected:Npn \@@_vline_ii:
6050 {
6051     \tl_clear:N \l_@@_tikz_rule_tl
6052     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```



```

6053 \bool_if:NTF \l_@@_dotted_bool
6054   \@@_vline_iv:
6055   {
6056     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6057     \@@_vline_iii:
6058     \@@_vline_v:
6059   }
6060 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6061 \cs_new_protected:Npn \@@_vline_iii:
6062 {
6063   \pgfpicture
6064   \pgfrememberpicturepositiononpagetrue
6065   \pgf@relevantforpicturesizefalse
6066   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6067   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6068   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6069   \dim_set:Nn \l_tmpb_dim
6070   {
6071     \pgf@x
6072     - 0.5 \l_@@_rule_width_dim
6073     +
6074     ( \arrayrulewidth * \l_@@_multiplicity_int
6075       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6076   }
6077   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6078   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6079   \bool_lazy_all:nT
6080   {
6081     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6082     { \cs_if_exist_p:N \CT@drsc@ }
6083     { ! \tl_if_blank_p:o \CT@drsc@ }
6084   }
6085   {
6086     \group_begin:
6087     \CT@drsc@
6088     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6089     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6090     \dim_set:Nn \l_@@_tmpd_dim
6091     {
6092       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6093       * ( \l_@@_multiplicity_int - 1 )
6094     }
6095     \pgfpathrectanglecorners
6096     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6097     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6098     \pgfusepath { fill }
6099     \group_end:
6100   }
6101   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6102   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6103   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6104   {
6105     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6106     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6107     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6108     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6109   }
6110   \CT@arc@
6111   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6112   \pgfsetrectcap
6113   \pgfusepathqstroke

```

```

6114 \endpgfpicture
6115 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6116 \cs_new_protected:Npn \@@_vline_iv:
6117 {
6118   \pgfpicture
6119   \pgfrememberpicturepositiononpagetrue
6120   \pgf@relevantforpicturesizefalse
6121   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6122   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6123   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6124   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6125   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6126   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6127   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6128   \CT@arc@
6129   \@@_draw_line:
6130   \endpgfpicture
6131 }

```

The following code is for the case when the user uses the key `tikz`.

```

6132 \cs_new_protected:Npn \@@_vline_v:
6133 {
6134   \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).

```

6135   \CT@arc@
6136   \tl_if_empty:NF \l_@@_rule_color_tl
6137   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6138   \pgfrememberpicturepositiononpagetrue
6139   \pgf@relevantforpicturesizefalse
6140   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6141   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6142   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6143   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6144   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6145   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6146   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6147   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6148   ( \l_tmpb_dim , \l_tmpa_dim ) --
6149   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6150   \end {tikzpicture }
6151 }

```

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).

```

6152 \cs_new_protected:Npn \@@_draw_vlines:
6153 {
6154   \int_step_inline:nnn
6155   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6156   {
6157     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6158     \c@jCol
6159     { \int_eval:n { \c@jCol + 1 } }
6160   }
6161   {
6162     \str_if_eq:eeF \l_@@_vlines_clist { all }
6163     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6164     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6165     }
6166 }

```

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}`.

```

6167 \cs_new_protected:Npn \@@_hline:n #1
6168 {

```

The group is for the options.

```

6169     \group_begin:
6170     \int_zero_new:N \l_@@_end_int
6171     \int_set_eq:NN \l_@@_end_int \c@jCol
6172     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6173     \@@_hline_i:
6174     \group_end:
6175 }

```

```

6176 \cs_new_protected:Npn \@@_hline_i:

```

```

6177 {
6178     \int_zero_new:N \l_@@_local_start_int
6179     \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`.

```

6180     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6181     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6182         \l_tmpb_tl
6183     {

```

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.

```

6184         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6185         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6186             { \@@_test_hline_in_block:nnnnn ##1 }
6187         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6188             { \@@_test_hline_in_block:nnnnn ##1 }
6189         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6190             { \@@_test_hline_in_stroken_block:nnnn ##1 }
6191         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6192         \bool_if:NTF \g_tmpa_bool
6193             {
6194                 \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.

```

6195             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6196         }
6197     {
6198         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6199             {
6200                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6201                 \@@_hline_ii:
6202                 \int_zero:N \l_@@_local_start_int
6203             }
6204     }
6205 }
6206 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6207     {
6208         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6209         \@@_hline_ii:
6210     }
6211 }

6212 \cs_new_protected:Npn \@@_test_in_corner_h:
6213 {
6214     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6215     {
6216         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6217         { \bool_set_false:N \g_tmpa_bool }
6218     }
6219     {
6220         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6221         {
6222             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6223             { \bool_set_false:N \g_tmpa_bool }
6224             {
6225                 \@@_if_in_corner:nT
6226                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6227                 { \bool_set_false:N \g_tmpa_bool }
6228             }
6229         }
6230     }
6231 }

6232 \cs_new_protected:Npn \@@_hline_ii:
6233 {
6234     \tl_clear:N \l_@@_tikz_rule_tl
6235     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6236     \bool_if:NTF \l_@@_dotted_bool
6237     \@@_hline_iv:
6238     {
6239         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6240         \@@_hline_iii:
6241         \@@_hline_v:
6242     }
6243 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6244 \cs_new_protected:Npn \@@_hline_iii:
6245 {
6246     \pgfpicture
6247     \pgfrememberpicturepositiononpagetrue
6248     \pgf@relevantforpicturesizefalse
6249     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6250     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6251     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6252     \dim_set:Nn \l_tmpb_dim
6253     {
6254         \pgf@y
6255         - 0.5 \l_@@_rule_width_dim
6256         +
6257         ( \arrayrulewidth * \l_@@_multiplicity_int
6258           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6259     }
6260     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6261     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6262     \bool_lazy_all:nT
6263     {

```

```

6264     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6265     { \cs_if_exist_p:N \CT@drsc@ }
6266     { ! \tl_if_blank_p:o \CT@drsc@ }
6267   }
6268   {
6269     \group_begin:
6270     \CT@drsc@
6271     \dim_set:Nn \l_@@_tmpd_dim
6272     {
6273       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6274       * ( \l_@@_multiplicity_int - 1 )
6275     }
6276     \pgfpathrectanglecorners
6277     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6278     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6279     \pgfusepathqfill
6280     \group_end:
6281   }
6282   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6283   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6284   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6285   {
6286     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6287     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6288     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6289     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6290   }
6291   \CT@arc@
6292   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6293   \pgfsetrectcap
6294   \pgfusepathqstroke
6295   \endpgfpicture
6296 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6297 \cs_new_protected:Npn \@@_hline_iv:
6298   {
6299     \pgfpicture
6300     \pgfrememberpicturepositiononpagetrue
6301     \pgf@relevantforpicturesizefalse
6302     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6303     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6304     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6305     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6306     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

6307 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6308 {
6309   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6310   \bool_if:NF \g_@@_delims_bool
6311   { \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.

```

6312   \tl_if_eq:NnF \g_@@_left_delim_tl (
6313     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6314   )
6315 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6316 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6317 \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6318 {
6319   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6320   \bool_if:NF \g_@@_delims_bool
6321   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6322   \tl_if_eq:NnF \g_@@_right_delim_tl )
6323   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6324 }
6325 \CT@arc@
6326 \@@_draw_line:
6327 \endpgfpicture
6328 }

```

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`).

```

6329 \cs_new_protected:Npn \@@_hline_v:
6330 {
6331   \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).

```

6332   \CT@arc@
6333   \tl_if_empty:NF \l_@@_rule_color_tl
6334   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6335   \pgfrememberpicturepositiononpagetrue
6336   \pgf@relevantforpicturesizefalse
6337   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6338   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6339   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6340   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6341   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6342   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6343   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6344   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6345   ( \l_tmpa_dim , \l_tmpb_dim ) --
6346   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6347   \end { tikzpicture }
6348 }

```

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).

```

6349 \cs_new_protected:Npn \@@_draw_hlines:
6350 {
6351   \int_step_inline:nnn
6352   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6353   {
6354     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6355     \c@iRow
6356     { \int_eval:n { \c@iRow + 1 } }
6357   }
6358   {
6359     \str_if_eq:eeF \l_@@_hlines_clist { all }
6360     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6361     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6362   }
6363 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6364 \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.

```

6365 \cs_set:Npn \@@_Hline_i:n #1
6366 {
6367   \peek_remove_spaces:n
6368   {
6369     \peek_meaning:NTF \Hline
6370     { \@@_Hline_ii:nn { #1 + 1 } }
6371     { \@@_Hline_iii:n { #1 } }
6372   }
6373 }
6374 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6375 \cs_set:Npn \@@_Hline_iii:n #1
6376 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6377 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6378 {
6379   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6380   \skip_vertical:N \l_@@_rule_width_dim
6381   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6382   {
6383     \@@_hline:n
6384     {
6385       multiplicity = #1 ,
6386       position = \int_eval:n { \c@iRow + 1 } ,
6387       total-width = \dim_use:N \l_@@_rule_width_dim ,
6388       #2
6389     }
6390   }
6391   \egroup
6392 }

```

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`).

```

6393 \cs_new_protected:Npn \@@_custom_line:n #1
6394 {
6395   \str_clear_new:N \l_@@_command_str
6396   \str_clear_new:N \l_@@_ccommand_str
6397   \str_clear_new:N \l_@@_letter_str
6398   \tl_clear_new:N \l_@@_other_keys_tl
6399   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6400 \bool_lazy_all:nTF
6401 {
6402   { \str_if_empty_p:N \l_@@_letter_str }
6403   { \str_if_empty_p:N \l_@@_command_str }
6404   { \str_if_empty_p:N \l_@@_ccommand_str }
6405 }
6406 { \@@_error:n { No~letter~and~no~command } }
6407 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6408 }
6409 \keys_define:nn { nicematrix / custom-line }
6410 {
6411   letter .str_set:N = \l_@@_letter_str ,
6412   letter .value_required:n = true ,
6413   command .str_set:N = \l_@@_command_str ,
6414   command .value_required:n = true ,
6415   ccommand .str_set:N = \l_@@_cccommand_str ,
6416   ccommand .value_required:n = true ,
6417 }
6418 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6419 \cs_new_protected:Npn \@@_custom_line_i:n #1
6420 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6421 \bool_set_false:N \l_@@_tikz_rule_bool
6422 \bool_set_false:N \l_@@_dotted_rule_bool
6423 \bool_set_false:N \l_@@_color_bool
6424 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6425 \bool_if:NT \l_@@_tikz_rule_bool
6426 {
6427   \IfPackageLoadedF { tikz }
6428   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6429   \bool_if:NT \l_@@_color_bool
6430   { \@@_error:n { color-in-custom-line-with-tikz } }
6431 }
6432 \bool_if:NT \l_@@_dotted_rule_bool
6433 {
6434   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6435   { \@@_error:n { key~multiplicity~with~dotted } }
6436 }
6437 \str_if_empty:NF \l_@@_letter_str
6438 {
6439   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6440   { \@@_error:n { Several~letters } }
6441   {
6442     \tl_if_in:NoTF
6443     \c_@@_forbidden_letters_str
6444     \l_@@_letter_str
6445     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6446   }

```

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.

```

6447 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6448 { \@@_v_custom_line:n { #1 } }
6449 }
6450 }
6451 }
6452 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6453 \str_if_empty:NF \l_@@_cccommand_str { \@@_c_custom_line:n { #1 } }
6454 }

```



```

6455 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6456 \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.

```

6457 \keys_define:nn { nicematrix / custom-line-bis }
6458 {
6459   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6460   multiplicity .initial:n = 1 ,
6461   multiplicity .value_required:n = true ,
6462   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6463   color .value_required:n = true ,
6464   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6465   tikz .value_required:n = true ,
6466   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6467   dotted .value_forbidden:n = true ,
6468   total-width .code:n = { } ,
6469   total-width .value_required:n = true ,
6470   width .code:n = { } ,
6471   width .value_required:n = true ,
6472   sep-color .code:n = { } ,
6473   sep-color .value_required:n = true ,
6474   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6475 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6476 \bool_new:N \l_@@_dotted_rule_bool
6477 \bool_new:N \l_@@_tikz_rule_bool
6478 \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`.

```

6479 \keys_define:nn { nicematrix / custom-line-width }
6480 {
6481   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6482   multiplicity .initial:n = 1 ,
6483   multiplicity .value_required:n = true ,
6484   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6485   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6486   \bool_set_true:N \l_@@_total_width_bool ,
6487   total-width .value_required:n = true ,
6488   width .meta:n = { total-width = #1 } ,
6489   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6490 }

```

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`).

```

6491 \cs_new_protected:Npn \@@_h_custom_line:n #1
6492 {

```

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`).

```

6493   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6494   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6495 }

```

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`).

```
6496 \cs_new_protected:Npn \@@_c_custom_line:n #1
6497 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6498   \exp_args:Nc \NewExpandableDocumentCommand
6499     { nicematrix - \l_@@_ccommand_str }
6500     { 0 { } m }
6501     {
6502       \noalign
6503       {
6504         \@@_compute_rule_width:n { #1 , ##1 }
6505         \skip_vertical:n { \l_@@_rule_width_dim }
6506         \clist_map_inline:nn
6507           { ##2 }
6508           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6509       }
6510     }
6511   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6512 }
```

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*.

```
6513 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6514 {
6515   \tl_if_in:nnTF { #2 } { - }
6516     { \@@_cut_on_hyphen:w #2 \q_stop }
6517     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6518   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6519     {
6520       \@@_hline:n
6521       {
6522         #1 ,
6523         start = \l_tmpa_tl ,
6524         end = \l_tmpb_tl ,
6525         position = \int_eval:n { \c@iRow + 1 } ,
6526         total-width = \dim_use:N \l_@@_rule_width_dim
6527       }
6528     }
6529 }

6530 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6531 {
6532   \bool_set_false:N \l_@@_tikz_rule_bool
6533   \bool_set_false:N \l_@@_total_width_bool
6534   \bool_set_false:N \l_@@_dotted_rule_bool
6535   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6536   \bool_if:NF \l_@@_total_width_bool
6537     {
6538       \bool_if:NTF \l_@@_dotted_rule_bool
6539         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6540         {
6541           \bool_if:NF \l_@@_tikz_rule_bool
6542             {
6543               \dim_set:Nn \l_@@_rule_width_dim
6544                 {
6545                   \arrayrulewidth * \l_@@_multiplicity_int
6546                   + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6547                 }
6548             }
6549         }
6550     }
6551 }
```

```

6552 \cs_new_protected:Npn \@@_v_custom_line:n #1
6553 {
6554   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6555   \tl_gput_right:Ne \g_@@_array_preamble_tl
6556     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6557   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6558     {
6559     \@@_vline:n
6560     {
6561     #1 ,
6562     position = \int_eval:n { \c@jCol + 1 } ,
6563     total-width = \dim_use:N \l_@@_rule_width_dim
6564     }
6565     }
6566   \@@_rec_preamble:n
6567 }

6568 \@@_custom_line:n
6569 { 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.

```

6570 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6571 {
6572   \int_compare:nNnT \l_tmpa_tl > { #1 }
6573   {
6574     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6575     {
6576       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6577       {
6578         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6579         { \bool_gset_false:N \g_tmpa_bool }
6580       }
6581     }
6582   }
6583 }

```

The same for vertical rules.

```

6584 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6585 {
6586   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6587   {
6588     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6589     {
6590       \int_compare:nNnT \l_tmpb_tl > { #2 }
6591       {
6592         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6593         { \bool_gset_false:N \g_tmpa_bool }
6594       }
6595     }
6596   }
6597 }

6598 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6599 {
6600   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6601   {
6602     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6603     {

```

```

6604         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6605         { \bool_gset_false:N \g_tmpa_bool }
6606         {
6607             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6608             { \bool_gset_false:N \g_tmpa_bool }
6609         }
6610     }
6611 }
6612 }
6613 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6614 {
6615     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6616     {
6617         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6618         {
6619             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6620             { \bool_gset_false:N \g_tmpa_bool }
6621             {
6622                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6623                 { \bool_gset_false:N \g_tmpa_bool }
6624             }
6625         }
6626     }
6627 }

```

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.

```

6628 \cs_new_protected:Npn \@@_compute_corners:
6629 {
6630     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6631     { \@@_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`).

```

6632     \clist_clear:N \l_@@_corners_cells_clist
6633     \clist_map_inline:Nn \l_@@_corners_cells_clist
6634     {
6635         \str_case:nnF { ##1 }
6636         {
6637             { NW }
6638             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6639             { NE }
6640             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6641             { SW }
6642             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6643             { SE }
6644             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6645         }
6646         { \@@_error:nn { bad-corner } { ##1 } }
6647     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6648     \clist_if_empty:NF \l_@@_corners_cells_clist
6649     {

```

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.

```

6650     \tl_gput_right:Np \g_@@_aux_tl
6651     {
6652         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6653         { \l_@@_corners_cells_clist }
6654     }
6655 }
6656 }

```

```

6657 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6658 {
6659     \int_step_inline:nnn { #1 } { #3 }
6660     {
6661         \int_step_inline:nnn { #2 } { #4 }
6662         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6663     }
6664 }

```

```

6665 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6666 {
6667     \cs_if_exist:cTF
6668     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6669     \prg_return_true:
6670     \prg_return_false:
6671 }

```

“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.

```

6672 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6673 {

```

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.

```

6674     \bool_set_false:N \l_tmpa_bool
6675     \int_zero_new:N \l_@@_last_empty_row_int
6676     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6677     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6678     {
6679         \bool_lazy_or:nnTF
6680         {
6681             \cs_if_exist_p:c
6682             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6683         }
6684         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6685         { \bool_set_true:N \l_tmpa_bool }
6686     }

```

```

6687         \bool_if:NF \l_tmpa_bool
6688         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6689     }
6690 }

```

Now, you determine the last empty cell in the row of number 1.

```

6691     \bool_set_false:N \l_tmpa_bool
6692     \int_zero_new:N \l_@@_last_empty_column_int
6693     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6694     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6695     {
6696         \bool_lazy_or:nnTF
6697         {
6698             \cs_if_exist_p:c
6699             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6700         }
6701         { \@@_if_in_block_p:nn { #1 } { ##1 } }
6702         { \bool_set_true:N \l_tmpa_bool }
6703     }
6704     \bool_if:NF \l_tmpa_bool
6705     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6706 }
6707 }

```

Now, we loop over the rows.

```

6708     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6709     {

```

We treat the row number ##1 with another loop.

```

6710         \bool_set_false:N \l_tmpa_bool
6711         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6712         {
6713             \bool_lazy_or:nnTF
6714             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6715             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6716             { \bool_set_true:N \l_tmpa_bool }
6717         }
6718         \bool_if:NF \l_tmpa_bool
6719         {
6720             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6721             \clist_put_right:Nn
6722             \l_@@_corners_cells_clist
6723             { ##1 - #####1 }
6724             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6725         }
6726     }
6727 }
6728 }
6729 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6730 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6731 \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:NeT \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.

```
6732 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6733 \keys_define:nn { nicematrix / NiceMatrixBlock }
6734 {
6735   auto-columns-width .code:n =
6736   {
6737     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6738     \dim_gzero_new:N \g_@@_max_cell_width_dim
6739     \bool_set_true:N \l_@@_auto_columns_width_bool
6740   }
6741 }
```

```
6742 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6743 {
6744   \int_gincr:N \g_@@_NiceMatrixBlock_int
6745   \dim_zero:N \l_@@_columns_width_dim
6746   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6747   \bool_if:NT \l_@@_block_auto_columns_width_bool
6748   {
6749     \cs_if_exist:cT
6750     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6751     {
6752       \dim_set:Nn \l_@@_columns_width_dim
6753       {
6754         \use:c
6755         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6756       }
6757     }
6758   }
6759 }
```

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`).

```
6760 {
6761   \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.

```
6762   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6763   {
6764     \bool_if:NT \l_@@_block_auto_columns_width_bool
6765     {
6766       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6767       \iow_shipout:Ne \@mainaux
6768       {
6769         \cs_gset:cpn
6770         { @@ _ 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.

```
6771         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6772       }
6773       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6774     }
6775   }
6776   \ignorespacesafterend
6777 }
```

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).

```

6778 \cs_new_protected:Npn \@@_create_extra_nodes:
6779 {
6780   \bool_if:nTF \l_@@_medium_nodes_bool
6781     {
6782       \bool_if:NTF \l_@@_large_nodes_bool
6783         \@@_create_medium_and_large_nodes:
6784         \@@_create_medium_nodes:
6785     }
6786     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6787 }

```

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`.

```

6788 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6789 {
6790   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6791   {
6792     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6793     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6794     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6795     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6796   }
6797   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6798   {
6799     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6800     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6801     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6802     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6803   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6804   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6805   {
6806     \int_step_variable:nnNn
6807     \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.

```

6808     {
6809         \cs_if_exist:cT
6810         { 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`.

```

6811     {
6812         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6813         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6814         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6815         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6816         {
6817             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6818             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6819         }

```

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`.

```

6820         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6821         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6822         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6823         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6824         {
6825             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6826             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6827         }
6828     }
6829 }
6830 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6831 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6832 {
6833     \dim_compare:nNnT
6834     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6835     {
6836         \@@_qpoint:n { row - \@@_i: - base }
6837         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6838         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6839     }
6840 }
6841 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6842 {
6843     \dim_compare:nNnT
6844     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6845     {
6846         \@@_qpoint:n { col - \@@_j: }
6847         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6848         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6849     }
6850 }
6851 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6852 \cs_new_protected:Npn \@@_create_medium_nodes:
6853 {
6854     \pgfpicture
6855     \pgfrememberpicturepositiononpagetrue
6856     \pgf@relevantforpicturesizefalse
6857     \@@_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”.

```

6858     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6859     \@@_create_nodes:
6860     \endpgfpicture
6861 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁴. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

6862 \cs_new_protected:Npn \@@_create_large_nodes:
6863 {
6864   \pgfpicture
6865   \pgfrememberpicturepositiononpagetrue
6866   \pgf@relevantforpicturesizefalse
6867   \@@_computations_for_medium_nodes:
6868   \@@_computations_for_large_nodes:
6869   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6870   \@@_create_nodes:
6871   \endpgfpicture
6872 }

6873 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6874 {
6875   \pgfpicture
6876   \pgfrememberpicturepositiononpagetrue
6877   \pgf@relevantforpicturesizefalse
6878   \@@_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”.

```

6879     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6880     \@@_create_nodes:
6881     \@@_computations_for_large_nodes:
6882     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6883     \@@_create_nodes:
6884     \endpgfpicture
6885 }

```

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.

```

6886 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6887 {
6888   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6889   \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`.

```

6890   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6891   {
6892     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6893     {
6894       (
6895         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6896         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6897       )
6898       / 2
6899     }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6900     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6901     { l_@@_row_\@@_i: _ min_dim }
6902   }
6903   \int_step_variable:nnN { \c@jCol - 1 } \@@_j:
6904   {
6905     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6906     {
6907       (
6908         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6909         \dim_use:c
6910         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6911       )
6912       / 2
6913     }
6914     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6915     { l_@@_column _ \@@_j: _ max _ dim }
6916   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6917   \dim_sub:cn
6918   { l_@@_column _ 1 _ min _ dim }
6919   \l_@@_left_margin_dim
6920   \dim_add:cn
6921   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6922   \l_@@_right_margin_dim
6923 }

```

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).

```

6924 \cs_new_protected:Npn \@@_create_nodes:
6925 {
6926   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6927   {
6928     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6929     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6930     \@@_pgf_rect_node:nnnnn
6931     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6932     { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6933     { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6934     { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6935     { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6936     \str_if_empty:NF \l_@@_name_str
6937     {
6938       \pgfnodealias
6939       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6940       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6941     }
6942   }
6943 }

```

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 .

```

6944   \seq_map_pairwise_function:NNN
6945   \g_@@_multicolumn_cells_seq
6946   \g_@@_multicolumn_sizes_seq
6947   \@@_node_for_multicolumn:nn
6948 }

```

```

6949 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6950 {
6951   \cs_set_nopar:Npn \@@_i: { #1 }
6952   \cs_set_nopar:Npn \@@_j: { #2 }
6953 }

```

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”).

```

6954 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6955 {
6956   \@@_extract_coords_values: #1 \q_stop
6957   \@@_pgf_rect_node:nnnnn
6958   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6959   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6960   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6961   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6962   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6963   \str_if_empty:NF \l_@@_name_str
6964   {
6965     \pgfnodealias
6966     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6967     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6968   }
6969 }

```

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).

```

6970 \keys_define:mn { nicematrix / Block / FirstPass }
6971 {
6972   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
6973     \bool_set_true:N \l_@@_p_block_bool ,
6974   j .value_forbidden:n = true ,
6975   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6976   l .value_forbidden:n = true ,
6977   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6978   r .value_forbidden:n = true ,
6979   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6980   c .value_forbidden:n = true ,
6981   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6982   L .value_forbidden:n = true ,
6983   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6984   R .value_forbidden:n = true ,
6985   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6986   C .value_forbidden:n = true ,
6987   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6988   t .value_forbidden:n = true ,
6989   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6990   T .value_forbidden:n = true ,
6991   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6992   b .value_forbidden:n = true ,
6993   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
6994   B .value_forbidden:n = true ,

```

```

6995 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
6996 m .value_forbidden:n = true ,
6997 v-center .meta:n = m ,
6998 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
6999 p .value_forbidden:n = true ,
7000 color .code:n =
7001   \@@_color:n { #1 }
7002   \tl_set_rescan:Nnn
7003     \l_@@_draw_tl
7004     { \char_set_catcode_other:N ! }
7005     { #1 } ,
7006 color .value_required:n = true ,
7007 respect-arraystretch .code:n =
7008   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7009 respect-arraystretch .value_forbidden:n = true ,
7010 }

```

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.

```

7011 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7012 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7013 {

```

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).

```

7014   \peek_remove_spaces:n
7015   {
7016     \tl_if_blank:nTF { #2 }
7017     { \@@_Block_ii:nnnn \c_one_int \c_one_int }
7018     {
7019       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7020       \@@_Block_i_czech \@@_Block_i
7021       #2 \q_stop
7022     }
7023     { #1 } { #3 } { #4 }
7024   }
7025 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7026 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnn { #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`.

```

7027 {
7028   \char_set_catcode_active:N -
7029   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnn { #1 } { #2 } }
7030 }

```

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.

```

7031 \cs_new_protected:Npn \@@_Block_ii:nnnn #1 #2 #3 #4 #5
7032 {

```

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).

```

7033   \bool_lazy_or:nnTF
7034     { \tl_if_blank_p:n { #1 } }
7035     { \str_if_eq_p:ee { * } { #1 } }
7036     { \int_set:Nn \l_tmpa_int { 100 } }
7037     { \int_set:Nn \l_tmpa_int { #1 } }
7038   \bool_lazy_or:nnTF
7039     { \tl_if_blank_p:n { #2 } }
7040     { \str_if_eq_p:ee { * } { #2 } }
7041     { \int_set:Nn \l_tmpb_int { 100 } }
7042     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7043   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7044     {
7045       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7046         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7047         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7048     }
7049     { \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.

```

7050   \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7051   \tl_set:Ne \l_tmpa_tl
7052     {
7053       { \int_use:N \c@iRow }
7054       { \int_use:N \c@jCol }
7055       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7056       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7057     }

```

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).

```

7058   \bool_set_false:N \l_tmpa_bool
7059   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7060     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7061   \bool_case:nF
7062     {
7063       \l_tmpa_bool                               { \@@_Block_vii:eennn }
7064       \l_@@_p_block_bool                         { \@@_Block_vii: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.

```

7065       \l_@@_X_bool                               { \@@_Block_v:eennn }
7066       { \tl_if_empty_p:n { #5 } }                 { \@@_Block_v:eennn }
7067       { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7068       { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7069     }
7070     { \@@_Block_v:eennn }
7071   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7072 }

```

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.

```

7073 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7074 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7075 {
7076   \int_gincr:N \g_@@_block_box_int
7077   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7078   {
7079     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7080     {
7081       \@@_actually_diagbox:nnnnnn
7082       { \int_use:N \c@iRow }
7083       { \int_use:N \c@jCol }
7084       { \int_eval:n { \c@iRow + #1 - 1 } }
7085       { \int_eval:n { \c@jCol + #2 - 1 } }
7086       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7087       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7088     }
7089   }
7090   \box_gclear_new:c
7091   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7092   \hbox_gset:cn
7093   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7094   {

```

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}`).

```

7095     \tl_if_empty:NTF \l_@@_color_tl
7096     { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7097     { \@@_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`.

```

7098     \int_compare:nNnT { #1 } = \c_one_int
7099     {
7100       \int_if_zero:nTF \c@iRow
7101       {

```

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}$

```

```

7102         \cs_set_eq:NN \Block \@@_NullBlock:
7103         \l_@@_code_for_first_row_tl
7104     }
7105     {
7106         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7107         {
7108             \cs_set_eq:NN \Block \@@_NullBlock:
7109             \l_@@_code_for_last_row_tl
7110         }
7111     }
7112     \g_@@_row_style_tl
7113 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7114     \@@_reset_arraystretch:
7115     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7116     #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.).

```

7117     \@@_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}`.

```

7118     \bool_if:NTF \l_@@_tabular_bool
7119     {
7120         \bool_lazy_all:nTF
7121         {
7122             { \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.

```

7123         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7124         { ! \g_@@_rotate_bool }
7125     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7126     {
7127         \use:e
7128     {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7129         \exp_not:N \begin { minipage }%
7130         [ \str_lowercase:o \l_@@_vpos_block_str ]
7131         { \l_@@_col_width_dim }
7132         \str_case:on \l_@@_hpos_block_str
7133         { c \centering r \raggedleft l \raggedright }
7134     }

```



```

7135         #5
7136         \end { minipage }
7137     }

```

In the other cases, we use a `{tabular}`.

```

7138     {
7139         \use:e
7140         {
7141             \exp_not:N \begin { tabular }%
7142             [ \str_lowercase:o \l_@@_vpos_block_str ]
7143             { @ { } \l_@@_hpos_block_str @ { } }
7144         }
7145         #5
7146         \end { tabular }
7147     }
7148 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7149     {
7150         \c_math_toggle_token
7151         \use:e
7152         {
7153             \exp_not:N \begin { array }%
7154             [ \str_lowercase:o \l_@@_vpos_block_str ]
7155             { @ { } \l_@@_hpos_block_str @ { } }
7156         }
7157         #5
7158         \end { array }
7159         \c_math_toggle_token
7160     }
7161 }

```

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).

```

7162     \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.

```

7163     \int_compare:nNnT { #2 } = \c_one_int
7164     {
7165         \dim_gset:Nn \g_@@_blocks_wd_dim
7166         {
7167             \dim_max:nn
7168             \g_@@_blocks_wd_dim
7169             {
7170                 \box_wd:c
7171                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7172             }
7173         }
7174     }

```

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.

```

7175     \bool_lazy_and:nnT
7176     { \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.

```

7177     { \str_if_empty_p:N \l_@@_vpos_block_str }
7178     {
7179         \dim_gset:Nn \g_@@_blocks_ht_dim

```

```

7180     {
7181         \dim_max:nn
7182         \g_@@_blocks_ht_dim
7183         {
7184             \box_ht:c
7185             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7186         }
7187     }
7188 \dim_gset:Nn \g_@@_blocks_dp_dim
7189 {
7190     \dim_max:nn
7191     \g_@@_blocks_dp_dim
7192     {
7193         \box_dp:c
7194         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7195     }
7196 }
7197 }
7198 \seq_gput_right:Ne \g_@@_blocks_seq
7199 {
7200     \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.

```

7201     {
7202         \exp_not:n { #3 } ,
7203         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7204     \bool_if:NT \g_@@_rotate_bool
7205     {
7206         \bool_if:NTF \g_@@_rotate_c_bool
7207         { m }
7208         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7209     }
7210 }
7211 {
7212     \box_use_drop:c
7213     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7214 }
7215 }
7216 \bool_set_false:N \g_@@_rotate_c_bool
7217 }

7218 \cs_new:Npn \@@_adjust_hpos_rotate:
7219 {
7220     \bool_if:NT \g_@@_rotate_bool
7221     {
7222         \str_set:Ne \l_@@_hpos_block_str
7223         {
7224             \bool_if:NTF \g_@@_rotate_c_bool
7225             { c }
7226             {
7227                 \str_case:onF \l_@@_vpos_block_str
7228                 { b l B l t r T r }
7229                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7230             }
7231         }
7232     }
7233 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7234 \cs_new_protected:Npn \@@_rotate_box_of_block:
7235 {
7236   \box_grotate:cn
7237   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7238   { 90 }
7239   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7240   {
7241     \vbox_gset_top:cn
7242     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7243     {
7244       \skip_vertical:n { 0.8 ex }
7245       \box_use:c
7246       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7247     }
7248   }
7249   \bool_if:NT \g_@@_rotate_c_bool
7250   {
7251     \hbox_gset:cn
7252     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7253     {
7254       \c_math_toggle_token
7255       \vcenter
7256       {
7257         \box_use:c
7258         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7259       }
7260       \c_math_toggle_token
7261     }
7262   }
7263 }

```

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.

```

7264 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7265 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7266 {
7267   \seq_gput_right:Ne \g_@@_blocks_seq
7268   {
7269     \l_tmpa_tl
7270     { \exp_not:n { #3 } }
7271     {
7272       \bool_if:NTF \l_@@_tabular_bool
7273       {
7274         \group_begin:

```

The following command will be no-op when respect-arraystretch is in force.

```

7275   \@@_reset_arraystretch:
7276   \exp_not:n
7277   {
7278     \dim_zero:N \extrarowheight
7279     #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).

```

7280         \bool_if:NT \c_@@_testphase_table_bool
7281         { \tag_stop:n { table } }
7282         \use:e
7283         {
7284             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7285             { @ { } \l_@@_hpos_block_str @ { } }
7286         }
7287         #5
7288         \end { tabular }
7289     }
7290 \group_end:
7291 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7292     {
7293         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7294         \@@_reset_arraystretch:
7295         \exp_not:n
7296         {
7297             \dim_zero:N \extrarowheight
7298             #4
7299             \c_math_toggle_token
7300             \use:e
7301             {
7302                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7303                 { @ { } \l_@@_hpos_block_str @ { } }
7304             }
7305             #5
7306             \end { array }
7307             \c_math_toggle_token
7308         }
7309         \group_end:
7310     }
7311 }
7312 }
7313 }

```

The following macro is for the case of a \Block which uses the key p.

```

7314 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7315 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7316 {
7317     \seq_gput_right:Ne \g_@@_blocks_seq
7318     {
7319         \l_tmpa_tl
7320         { \exp_not:n { #3 } }
7321         {
7322             \group_begin:
7323             \exp_not:n { #4 #5 }
7324             \group_end:
7325         }
7326     }
7327 }

```

The following macro is for the case of a \Block which uses the key p.

```

7328 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7329 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7330 {
7331     \seq_gput_right:Ne \g_@@_blocks_seq
7332     {

```

```

7333     \l_tmpa_tl
7334     { \exp_not:n { #3 } }
7335     { \exp_not:n { #4 #5 } }
7336   }
7337 }

```

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).

```

7338 \keys_define:nn { nicematrix / Block / SecondPass }
7339 {
7340   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7341   ampersand-in-blocks .default:n = true ,
7342   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7343   tikz .code:n =
7344     \IfPackageLoadedTF { tikz }
7345     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7346     { \@@_error:n { tikz~key~without~tikz } } ,
7347   tikz .value_required:n = true ,
7348   fill .code:n =
7349     \tl_set_rescan:Nnn
7350     \l_@@_fill_tl
7351     { \char_set_catcode_other:N ! }
7352     { #1 } ,
7353   fill .value_required:n = true ,
7354   opacity .tl_set:N = \l_@@_opacity_tl ,
7355   opacity .value_required:n = true ,
7356   draw .code:n =
7357     \tl_set_rescan:Nnn
7358     \l_@@_draw_tl
7359     { \char_set_catcode_other:N ! }
7360     { #1 } ,
7361   draw .default:n = default ,
7362   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7363   rounded-corners .default:n = 4 pt ,
7364   color .code:n =
7365     \@@_color:n { #1 }
7366     \tl_set_rescan:Nnn
7367     \l_@@_draw_tl
7368     { \char_set_catcode_other:N ! }
7369     { #1 } ,
7370   borders .clist_set:N = \l_@@_borders_clist ,
7371   borders .value_required:n = true ,
7372   hvlines .meta:n = { vlines , hlines } ,
7373   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7374   vlines .default:n = true ,
7375   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7376   hlines .default:n = true ,
7377   line-width .dim_set:N = \l_@@_line_width_dim ,
7378   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7379   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7380     \bool_set_true:N \l_@@_p_block_bool ,
7381   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7382   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7383   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7384   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7385     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7386   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7387     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,

```

```

7388 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7389         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7390 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7391 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7392 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7393 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7394 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7395 m .value_forbidden:n = true ,
7396 v-center .meta:n = m ,
7397 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7398 p .value_forbidden:n = true ,
7399 name .tl_set:N = \l_@@_block_name_str ,
7400 name .value_required:n = true ,
7401 name .initial:n = ,
7402 respect-arraystretch .code:n =
7403     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7404 respect-arraystretch .value_forbidden:n = true ,
7405 transparent .bool_set:N = \l_@@_transparent_bool ,
7406 transparent .default:n = true ,
7407 transparent .initial:n = false ,
7408 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7409 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7410 \cs_new_protected:Npn \@@_draw_blocks:
7411 {
7412     \bool_if:NTF \c_@@_tagging_array_bool
7413     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7414     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7415     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7416 }
7417 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7418 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7419 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7420     \int_zero_new:N \l_@@_last_row_int
7421     \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

7422     \int_compare:nNnTF { #3 } > { 99 }
7423     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7424     { \int_set:Nn \l_@@_last_row_int { #3 } }
7425     \int_compare:nNnTF { #4 } > { 99 }
7426     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7427     { \int_set:Nn \l_@@_last_col_int { #4 } }
7428     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7429     {
7430         \bool_lazy_and:nnTF
7431         \l_@@_preamble_bool
7432         {
7433             \int_compare_p:n
7434             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7435         }

```

```

7436     {
7437     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7438     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7439     \@@_msg_redirect_name:nn { columns-not-used } { none }
7440     }
7441     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7442   }
7443   {
7444   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7445   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7446   {
7447     \@@_Block_v:nneenn
7448     { #1 }
7449     { #2 }
7450     { \int_use:N \l_@@_last_row_int }
7451     { \int_use:N \l_@@_last_col_int }
7452     { #5 }
7453     { #6 }
7454   }
7455 }
7456 }

```

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

```

7457 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7458 {

```

The group is for the keys.

```

7459   \group_begin:
7460   \int_compare:nNnT { #1 } = { #3 }
7461   { \str_set:Nn \l_@@_vpos_block_str { t } }
7462   \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`.

```

7463   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7464   \bool_lazy_and:nnT
7465   \l_@@_vlines_block_bool
7466   { ! \l_@@_ampersand_bool }
7467   {
7468     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7469     {
7470       \@@_vlines_block:nnn
7471       { \exp_not:n { #5 } }
7472       { #1 - #2 }
7473       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7474     }
7475   }
7476   \bool_if:NT \l_@@_hlines_block_bool
7477   {
7478     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7479     {
7480       \@@_hlines_block:nnn
7481       { \exp_not:n { #5 } }
7482       { #1 - #2 }
7483       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7484     }
7485   }
7486   \bool_if:NF \l_@@_transparent_bool
7487   {
7488     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7489     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7490     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7491     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7492   }
7493 }

```

```

7494 \tl_if_empty:NF \l_@@_draw_tl
7495 {
7496   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7497   { \@@_error:n { hlines~with~color } }
7498   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7499   {
7500     \@@_stroke_block:nnn

```

#5 are the options

```

7501     { \exp_not:n { #5 } }
7502     { #1 - #2 }
7503     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7504   }
7505   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7506   { { #1 } { #2 } { #3 } { #4 } }
7507 }

7508 \clist_if_empty:NF \l_@@_borders_clist
7509 {
7510   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7511   {
7512     \@@_stroke_borders_block:nnn
7513     { \exp_not:n { #5 } }
7514     { #1 - #2 }
7515     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7516   }
7517 }

7518 \tl_if_empty:NF \l_@@_fill_tl
7519 {
7520   \tl_if_empty:NF \l_@@_opacity_tl
7521   {
7522     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7523       {
7524         \tl_set:Ne \l_@@_fill_tl
7525         {
7526           [ opacity = \l_@@_opacity_tl ,
7527             \tl_tail:o \l_@@_fill_tl
7528         ]
7529       }
7530     ]
7531     \tl_set:Ne \l_@@_fill_tl
7532     { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7533   }
7534 }

7535 \tl_gput_right:Ne \g_@@_pre_code_before_tl
7536 {
7537   \exp_not:N \roundedrectanglecolor
7538   \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7539     { \l_@@_fill_tl }
7540     { { \l_@@_fill_tl } }
7541     { #1 - #2 }
7542     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7543     { \dim_use:N \l_@@_rounded_corners_dim }
7544   ]
7545 }

```



```

7546 \seq_if_empty:NF \l_@@_tikz_seq
7547 {
7548   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7549   {
7550     \@@_block_tikz:nnnnn
7551     { \seq_use:Nn \l_@@_tikz_seq { , } }
7552     { #1 }
7553     { #2 }
7554     { \int_use:N \l_@@_last_row_int }
7555     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7556   }
7557 }

7558 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7559 {
7560   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7561   {
7562     \@@_actually_diagbox:nnnnnn
7563     { #1 }
7564     { #2 }
7565     { \int_use:N \l_@@_last_row_int }
7566     { \int_use:N \l_@@_last_col_int }
7567     { \exp_not:n { ##1 } }
7568     { \exp_not:n { ##2 } }
7569   }
7570 }

```

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	two
six	five
seven	eight

The construction of the node corresponding to the merged cells.

```

7571 \pgfpicture
7572 \pgfrememberpicturepositiononpagetrue
7573 \pgf@relevantforpicturesizefalse
7574 \@@_qpoint:n { row - #1 }
7575 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7576 \@@_qpoint:n { col - #2 }
7577 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7578 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7579 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7580 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7581 \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.

```

7582 \@@_pgf_rect_node:nnnnn
7583   { \@@_env: - #1 - #2 - block }
7584   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7585 \str_if_empty:NF \l_@@_block_name_str
7586   {
7587     \pgfnodealias
7588     { \@@_env: - \l_@@_block_name_str }
7589     { \@@_env: - #1 - #2 - block }
7590     \str_if_empty:NF \l_@@_name_str
7591     {
7592       \pgfnodealias
7593       { \l_@@_name_str - \l_@@_block_name_str }
7594       { \@@_env: - #1 - #2 - block }
7595     }
7596   }

```

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.

```

7597 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7598   {
7599     \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.

```

7600 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7601   {

```

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.

```

7602     \cs_if_exist:cT
7603     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7604     {
7605       \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7606       {
7607         \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7608         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7609       }
7610     }
7611   }

```

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.

```

7612     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7613     {
7614       \@@_qpoint:n { col - #2 }
7615       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7616     }
7617     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7618     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7619     {
7620       \cs_if_exist:cT
7621       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7622       {
7623         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7624         {
7625           \pgfpointanchor
7626           { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7627           { east }

```

```

7628         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7629     }
7630 }
7631 }
7632 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7633 {
7634     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7635     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7636 }
7637 \@@_pgf_rect_node:nnnnn
7638 { \@@_env: - #1 - #2 - block - short }
7639     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7640 }

```

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.

```

7641 \bool_if:NT \l_@@_medium_nodes_bool
7642 {
7643     \@@_pgf_rect_node:nnn
7644     { \@@_env: - #1 - #2 - block - medium }
7645     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7646     {
7647         \pgfpointanchor
7648         { \@@_env:
7649             - \int_use:N \l_@@_last_row_int
7650             - \int_use:N \l_@@_last_col_int - medium
7651         }
7652         { south~east }
7653     }
7654 }
7655 \endpgfpicture

```

```

7656 \bool_if:NTF \l_@@_ampersand_bool
7657 {
7658     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7659     \int_zero_new:N \l_@@_split_int
7660     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7661     \pgfpicture
7662     \pgfrememberpicturepositiononpagetrue
7663     \pgf@relevantforpicturesizefalse
7664
7665     \@@_qpoint:n { row - #1 }
7666     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7667     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7668     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7669     \@@_qpoint:n { col - #2 }
7670     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7671     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7672     \dim_set:Nn \l_tmpb_dim
7673     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7674     \bool_lazy_or:nnT
7675     \l_@@_vlines_block_bool
7676     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7677     {
7678         \int_step_inline:nn { \l_@@_split_int - 1 }
7679         {
7680             \pgfpathmoveto
7681             {
7682                 \pgfpoint
7683                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7684                 \l_@@_tmpc_dim
7685             }
7686             \pgfpathlineto

```

```

7687         {
7688             \pgfpoint
7689             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7690             \l_@@_tmpd_dim
7691         }
7692         \CT@arc@
7693         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7694         \pgfsetrectcap
7695         \pgfusepathqstroke
7696     }
7697 }
7698 \@@_qpoint:n { row - #1 - base }
7699 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7700 \int_step_inline:nn \l_@@_split_int
7701 {
7702     \group_begin:
7703     \dim_set:Nn \col@sep
7704     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7705     \pgftransformshift
7706     {
7707         \pgfpoint
7708         {
7709             \l_tmpa_dim + ##1 \l_tmpb_dim -
7710             \str_case:on \l_@@_hpos_block_str
7711             {
7712                 l { \l_tmpb_dim + \col@sep }
7713                 c { 0.5 \l_tmpb_dim }
7714                 r { \col@sep }
7715             }
7716         }
7717         { \l_@@_tmpc_dim }
7718     }
7719     \pgfset { inner~sep = \c_zero_dim }
7720     \pgfnode
7721     { rectangle }
7722     {
7723         \str_case:on \l_@@_hpos_block_str
7724         {
7725             c { base }
7726             l { base-west }
7727             r { base-east }
7728         }
7729     }
7730     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7731     \group_end:
7732 }
7733 \endpgfpicture
7734 }

```

Now the case where there is no ampersand & in the content of the block.

```

7735 {
7736     \bool_if:NTF \l_@@_p_block_bool
7737     {

```

When the final user has used the key p, we have to compute the width.

```

7738     \pgfpicture
7739     \pgfrememberpicturepositiononpagetrue
7740     \pgf@relevantforpicturesizefalse
7741     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7742     {
7743         \@@_qpoint:n { col - #2 }
7744         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7745         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7746     }

```

```

7747     {
7748         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7749         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7750         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7751     }
7752     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7753 \endpgfpicture
7754 \hbox_set:Nn \l_@@_cell_box
7755 {
7756     \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7757         { \g_tmpb_dim }
7758         \str_case:on \l_@@_hpos_block_str
7759             { c \centering r \raggedleft l \raggedright j { } }
7760         #6
7761     \end { minipage }
7762 }
7763 }
7764 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7765 \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.

```

7766 \pgfpicture
7767 \pgfrememberpicturepositiononpagetrue
7768 \pgf@relevantforpicturesizefalse
7769 \bool_lazy_any:nTF
7770 {
7771     { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7772     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7773     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7774     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7775 }
7776 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7777     \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`.

```

7778     \bool_if:nT \g_@@_last_col_found_bool
7779     {
7780         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7781         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7782     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7783     \tl_set:Ne \l_tmpa_tl
7784     {
7785         \str_case:on \l_@@_vpos_block_str
7786         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7787         { } { % added 2024-06-29
7788             \str_case:on \l_@@_hpos_block_str
7789             {
7790                 c { center }
7791                 l { west }
7792                 r { east }
7793                 j { center }
7794             }
7795         }
7796     c {
7797         \str_case:on \l_@@_hpos_block_str

```

```

7798         {
7799             c { center }
7800             l { west }
7801             r { east }
7802             j { center }
7803         }
7804
7805     }
7806     T {
7807         \str_case:on \l_@@_hpos_block_str
7808         {
7809             c { north }
7810             l { north-west }
7811             r { north-east }
7812             j { north }
7813         }
7814
7815     }
7816     B {
7817         \str_case:on \l_@@_hpos_block_str
7818         {
7819             c { south }
7820             l { south-west }
7821             r { south-east }
7822             j { south }
7823         }
7824
7825     }
7826 }
7827
7828 \pgftransformshift
7829 {
7830     \pgfpointanchor
7831     {
7832         \@@_env: - #1 - #2 - block
7833         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7834     }
7835     { \l_tmpa_tl }
7836 }
7837 \pgfset { inner-sep = \c_zero_dim }
7838 \pgfnode
7839 { rectangle }
7840 { \l_tmpa_tl }
7841 { \box_use_drop:N \l_@@_cell_box } { } { }
7842 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7843 {
7844     \pgfextracty \l_tmpa_dim
7845     {
7846         \@@_qpoint:n
7847         {
7848             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7849             - base
7850         }
7851     }
7852     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

7853 \pgfpointanchor
7854 {
7855     \@@_env: - #1 - #2 - block
7856     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }

```

```

7857     }
7858     {
7859         \str_case:on \l_@@_hpos_block_str
7860         {
7861             c { center }
7862             l { west }
7863             r { east }
7864             j { center }
7865         }
7866     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7867         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7868         \pgfset { inner~sep = \c_zero_dim }
7869         \pgfnode
7870         { rectangle }
7871         {
7872             \str_case:on \l_@@_hpos_block_str
7873             {
7874                 c { base }
7875                 l { base~west }
7876                 r { base~east }
7877                 j { base }
7878             }
7879         }
7880         { \box_use_drop:N \l_@@_cell_box } { } { }
7881     }
7882     \endpgfpicture
7883 }
7884 \group_end:
7885 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

7886 \cs_set_protected:Npn \@@_fill:nnnn #1 #2 #3 #4 #5
7887 {
7888     \pgfpicture
7889     \pgfrememberpicturepositiononpagetrue
7890     \pgf@relevantforpicturesizefalse
7891     \pgfpathrectanglecorners
7892     { \pgfpoint { #2 } { #3 } }
7893     { \pgfpoint { #4 } { #5 } }
7894     \pgfsetfillcolor { #1 }
7895     \pgfusepath { fill }
7896     \endpgfpicture
7897 }

```

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).

```

7898 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7899 {
7900     \group_begin:
7901     \tl_clear:N \l_@@_draw_tl
7902     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7903     \keys_set_known:nm { nicematrix / BlockStroke } { #1 }
7904     \pgfpicture
7905     \pgfrememberpicturepositiononpagetrue
7906     \pgf@relevantforpicturesizefalse
7907     \tl_if_empty:NF \l_@@_draw_tl
7908     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7909     \tl_if_eq:NnTF \l_@@_draw_tl { default }
7910         { \CT@arc@ }
7911         { \@@_color:o \l_@@_draw_tl }
7912     }
7913     \pgfsetcornersarced
7914     {
7915         \pgfpoint
7916         { \l_@@_rounded_corners_dim }
7917         { \l_@@_rounded_corners_dim }
7918     }
7919     \@@_cut_on_hyphen:w #2 \q_stop
7920     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7921     {
7922         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7923         {
7924             \@@_qpoint:n { row - \l_tmpa_tl }
7925             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7926             \@@_qpoint:n { col - \l_tmpb_tl }
7927             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7928             \@@_cut_on_hyphen:w #3 \q_stop
7929             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7930                 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7931             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7932                 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7933             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7934             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7935             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7936             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7937             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7938             \pgfpathrectanglecorners
7939                 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7940                 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7941             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7942                 { \pgfusepathqstroke }
7943                 { \pgfusepath { stroke } }
7944         }
7945     }
7946     \endpgfpicture
7947     \group_end:
7948 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7949 \keys_define:nn { nicematrix / BlockStroke }
7950 {
7951     color .tl_set:N = \l_@@_draw_tl ,
7952     draw .code:n =
7953         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7954     draw .default:n = default ,
7955     line-width .dim_set:N = \l_@@_line_width_dim ,
7956     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7957     rounded-corners .default:n = 4 pt
7958 }

```

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).

```

7959 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7960 {
7961     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7962     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7963     \@@_cut_on_hyphen:w #2 \q_stop

```



```

7964 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7965 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7966 \@@_cut_on_hyphen:w #3 \q_stop
7967 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7968 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7969 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7970 {
7971   \use:e
7972   {
7973     \@@_vline:n
7974     {
7975       position = ##1 ,
7976       start = \l_@@_tmpc_tl ,
7977       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7978       total-width = \dim_use:N \l_@@_line_width_dim
7979     }
7980   }
7981 }
7982 }
7983 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7984 {
7985   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7986   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7987   \@@_cut_on_hyphen:w #2 \q_stop
7988   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7989   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7990   \@@_cut_on_hyphen:w #3 \q_stop
7991   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7992   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7993   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7994   {
7995     \use:e
7996     {
7997       \@@_hline:n
7998       {
7999         position = ##1 ,
8000         start = \l_@@_tmpd_tl ,
8001         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8002         total-width = \dim_use:N \l_@@_line_width_dim
8003       }
8004     }
8005   }
8006 }

```

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).

```

8007 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8008 {
8009   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8010   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8011   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8012     { \@@_error:n { borders~forbidden } }
8013     {
8014       \tl_clear_new:N \l_@@_borders_tikz_tl
8015       \keys_set:no
8016         { nicematrix / OnlyForTikzInBorders }
8017       \l_@@_borders_clist
8018       \@@_cut_on_hyphen:w #2 \q_stop
8019       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8020       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8021       \@@_cut_on_hyphen:w #3 \q_stop
8022       \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }

```

```

8023     \tl_set:Np \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8024     \@@_stroke_borders_block_i:
8025   }
8026 }

8027 \hook_gput_code:nnn { begindocument } { . }
8028 {
8029   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8030   {
8031     \c_@@_pgfortikzpicture_tl
8032     \@@_stroke_borders_block_ii:
8033     \c_@@_endpgfortikzpicture_tl
8034   }
8035 }

8036 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8037 {
8038   \pgfrememberpicturepositiononpagetrue
8039   \pgf@relevantforpicturesizefalse
8040   \CT@arc@
8041   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8042   \clist_if_in:NnT \l_@@_borders_clist { right }
8043   { \@@_stroke_vertical:n \l_tmpb_tl }
8044   \clist_if_in:NnT \l_@@_borders_clist { left }
8045   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8046   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8047   { \@@_stroke_horizontal:n \l_tmpa_tl }
8048   \clist_if_in:NnT \l_@@_borders_clist { top }
8049   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8050 }

8051 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8052 {
8053   tikz .code:n =
8054     \cs_if_exist:NTF \tikzpicture
8055     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8056     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8057   tikz .value_required:n = true ,
8058   top .code:n = ,
8059   bottom .code:n = ,
8060   left .code:n = ,
8061   right .code:n = ,
8062   unknown .code:n = \@@_error:n { bad~border }
8063 }

```

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).

```

8064 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8065 {
8066   \@@_qpoint:n \l_@@_tmpc_tl
8067   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8068   \@@_qpoint:n \l_tmpa_tl
8069   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8070   \@@_qpoint:n { #1 }
8071   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8072   {
8073     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8074     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8075     \pgfusepathqstroke
8076   }
8077   {
8078     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8079     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8080   }
8081 }

```

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).

```

8082 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8083 {
8084   \@@_qpoint:n \l_@@_tmpd_tl
8085   \clist_if_in:NnTF \l_@@_borders_clist { left }
8086     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8087     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8088   \@@_qpoint:n \l_tmpb_tl
8089   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8090   \@@_qpoint:n { #1 }
8091   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8092     {
8093       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8094       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8095       \pgfusepathqstroke
8096     }
8097     {
8098       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8099         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8100     }
8101 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8102 \keys_define:nn { nicematrix / BlockBorders }
8103 {
8104   borders .clist_set:N = \l_@@_borders_clist ,
8105   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8106   rounded-corners .default:n = 4 pt ,
8107   line-width .dim_set:N = \l_@@_line_width_dim
8108 }

```

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.

```

8109 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8110 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8111 {
8112   \begin { tikzpicture }
8113   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8114   \clist_map_inline:nn { #1 }
8115   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8116     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8117     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8118     (
8119       [
8120         xshift = \dim_use:N \l_@@_offset_dim ,
8121         yshift = - \dim_use:N \l_@@_offset_dim
8122       ]
8123       #2 -| #3
8124     )
8125     rectangle
8126     (
8127     [

```

```

8128         xshift = - \dim_use:N \l_@@_offset_dim ,
8129         yshift = \dim_use:N \l_@@_offset_dim
8130     ]
8131     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8132     ) ;
8133 }
8134 \end { tikzpicture }
8135 }

8136 \keys_define:nn { nicematrix / SpecialOffset }
8137 { 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.

```

8138 \cs_new_protected:Npn \@@_NullBlock:
8139 { \@@_collect_options:n { \@@_NullBlock_i: } }
8140 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8141 { }

```

27 How to draw the dotted lines transparently

```

8142 \cs_set_protected:Npn \@@_renew_matrix:
8143 {
8144     \RenewDocumentEnvironment { pmatrix } { }
8145     { \pNiceMatrix }
8146     { \endpNiceMatrix }
8147     \RenewDocumentEnvironment { vmatrix } { }
8148     { \vNiceMatrix }
8149     { \endvNiceMatrix }
8150     \RenewDocumentEnvironment { Vmatrix } { }
8151     { \VNiceMatrix }
8152     { \endVNiceMatrix }
8153     \RenewDocumentEnvironment { bmatrix } { }
8154     { \bNiceMatrix }
8155     { \endbNiceMatrix }
8156     \RenewDocumentEnvironment { Bmatrix } { }
8157     { \BNiceMatrix }
8158     { \endBNiceMatrix }
8159 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8160 \keys_define:nn { nicematrix / Auto }
8161 {
8162     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8163     columns-type .value_required:n = true ,
8164     l .meta:n = { columns-type = l } ,
8165     r .meta:n = { columns-type = r } ,
8166     c .meta:n = { columns-type = c } ,
8167     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8168     delimiters / color .value_required:n = true ,
8169     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8170     delimiters / max-width .default:n = true ,
8171     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8172     delimiters .value_required:n = true ,

```

```

8173     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8174     rounded-corners .default:n = 4 pt
8175 }

8176 \NewDocumentCommand \AutoNiceMatrixWithDelims
8177 { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8178 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8179 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8180 {

```

The group is for the protection of the keys.

```

8181     \group_begin:
8182     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8183     \use:e
8184     {
8185         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8186         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8187         [ \exp_not:o \l_tmpa_tl ]
8188     }
8189     \int_if_zero:nT \l_@@_first_row_int
8190     {
8191         \int_if_zero:nT \l_@@_first_col_int { & }
8192         \prg_replicate:nn { #4 - 1 } { & }
8193         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8194     }
8195     \prg_replicate:mn { #3 }
8196     {
8197         \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`).

```

8198         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8199         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8200     }
8201     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8202     {
8203         \int_if_zero:nT \l_@@_first_col_int { & }
8204         \prg_replicate:nn { #4 - 1 } { & }
8205         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8206     }
8207     \end { NiceArrayWithDelims }
8208     \group_end:
8209 }

8210 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8211 {
8212     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8213     {
8214         \bool_gset_true:N \g_@@_delims_bool
8215         \str_gset_Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8216         \AutoNiceMatrixWithDelims { #2 } { #3 }
8217     }
8218 }

8219 \@@_define_com:nnn p ( )
8220 \@@_define_com:nnn b [ ]
8221 \@@_define_com:nnn v | |
8222 \@@_define_com:nnn V \! \!
8223 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8224 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8225 {

```

```

8226 \group_begin:
8227 \bool_gset_false:N \g_@@_delims_bool
8228 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8229 \group_end:
8230 }

```

29 The redefinition of the command `\dotfill`

```

8231 \cs_set_eq:NN \@@_old_dotfill \dotfill
8232 \cs_new_protected:Npn \@@_dotfill:
8233 {

```

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}`).

```

8234 \@@_old_dotfill
8235 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8236 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8237 \cs_new_protected:Npn \@@_dotfill_i:
8238 { \dim_compare:nNnT { \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.

```

8239 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8240 {
8241 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8242 {
8243 \@@_actually_diagbox:nnnnnn
8244 { \int_use:N \c@iRow }
8245 { \int_use:N \c@jCol }
8246 { \int_use:N \c@iRow }
8247 { \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.

```

8248 { \g_@@_row_style_tl \exp_not:n { #1 } }
8249 { \g_@@_row_style_tl \exp_not:n { #2 } }
8250 }

```

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`.

```

8251 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8252 {
8253 { \int_use:N \c@iRow }
8254 { \int_use:N \c@jCol }
8255 { \int_use:N \c@iRow }
8256 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8257     { }
8258   }
8259 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnn` 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.

```

8260 \cs_new_protected:Npn \@@_actually_diagbox:nnnnn #1 #2 #3 #4 #5 #6
8261 {
8262   \pgfpicture
8263   \pgf@relevantforpicturesizefalse
8264   \pgfrememberpicturepositiononpagetrue
8265   \@@_qpoint:n { row - #1 }
8266   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8267   \@@_qpoint:n { col - #2 }
8268   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8269   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8270   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8271   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8272   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8273   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8274   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8275   {

```

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.

```

8276     \CT@arc@
8277     \pgfsetroundcap
8278     \pgfusepathqstroke
8279   }
8280   \pgfset { inner~sep = 1 pt }
8281   \pgfscope
8282   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8283   \pgfnode { rectangle } { south~west }
8284   {
8285     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8286     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8287     \end { minipage }
8288   }
8289   { }
8290   { }
8291 \endpgfscope
8292 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8293 \pgfnode { rectangle } { north~east }
8294 {
8295   \begin { minipage } { 20 cm }
8296   \raggedleft
8297   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8298   \end { minipage }
8299 }
8300 { }
8301 { }
8302 \endpgfpicture
8303 }

```

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. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
8304 \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 `\`.

```
8305 \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`.

```
8306 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8307 {
8308   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8309   \@@_CodeAfter_iv:n
8310 }
```

We catch the argument of the command `\end` (in `#1`).

```
8311 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8312 {
```

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.

```
8313   \str_if_eq:eeTF \@currenvir { #1 }
8314   { \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`.

```
8315   {
8316     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8317     \@@_CodeAfter_ii:n
8318   }
8319 }
```

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 column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8320 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8321 {
8322   \pgfpicture
8323   \pgfrememberpicturepositiononpagetrue
8324   \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.

```

8325 \@@_qpoint:n { row - 1 }
8326 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8327 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8328 \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).

```

8329 \bool_if:nTF { #3 }
8330 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8331 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8332 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8333 {
8334   \cs_if_exist:cT
8335   { \pgf @ sh @ ns @ \@@_env: - #1 - #2 }
8336   {
8337     \pgfpointanchor
8338     { \@@_env: - #1 - #2 }
8339     { \bool_if:nTF { #3 } { west } { east } }
8340     \dim_set:Nn \l_tmpa_dim
8341     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8342   }
8343 }

```

Now we can put the delimiter with a node of PGF.

```

8344 \pgfset { inner~sep = \c_zero_dim }
8345 \dim_zero:N \nulldelimiterspace
8346 \pgftransformshift
8347 {
8348   \pgfpoint
8349   { \l_tmpa_dim }
8350   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8351 }
8352 \pgfnode
8353 { rectangle }
8354 { \bool_if:nTF { #3 } { east } { west } }
8355 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8356 \nullfont
8357 \c_math_toggle_token
8358 \@@_color:o \l_@@_delimiters_color_tl
8359 \bool_if:nTF { #3 } { \left #1 } { \left . }
8360 \vcenter
8361 {
8362   \nullfont
8363   \hrule \@height
8364   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8365   \@depth \c_zero_dim
8366   \@width \c_zero_dim
8367 }
8368 \bool_if:nTF { #3 } { \right . } { \right #1 }
8369 \c_math_toggle_token
8370 }
8371 { }
8372 { }
8373 \endpgfpicture
8374 }

```

33 The command `\SubMatrix`

```

8375 \keys_define:nn { nicematrix / sub-matrix }
8376 {
8377   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8378   extra-height .value_required:n = true ,
8379   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8380   left-xshift .value_required:n = true ,
8381   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8382   right-xshift .value_required:n = true ,
8383   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8384   xshift .value_required:n = true ,
8385   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8386   delimiters / color .value_required:n = true ,
8387   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8388   slim .default:n = true ,
8389   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8390   hlines .default:n = all ,
8391   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8392   vlines .default:n = all ,
8393   hvlines .meta:n = { hlines, vlines } ,
8394   hvlines .value_forbidden:n = true
8395 }
8396 \keys_define:nn { nicematrix }
8397 {
8398   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8399   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8400   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8401   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8402 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8403 \keys_define:nn { nicematrix / SubMatrix }
8404 {
8405   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8406   delimiters / color .value_required:n = true ,
8407   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8408   hlines .default:n = all ,
8409   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8410   vlines .default:n = all ,
8411   hvlines .meta:n = { hlines, vlines } ,
8412   hvlines .value_forbidden:n = true ,
8413   name .code:n =
8414     \tl_if_empty:nTF { #1 }
8415     { \@@_error:n { Invalid-name } }
8416     {
8417       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8418       {
8419         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8420         { \@@_error:nn { Duplicate-name~for~SubMatrix } { #1 } }
8421         {
8422           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8423           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8424         }
8425       }
8426     } \@@_error:n { Invalid-name } }
8427   } ,
8428   name .value_required:n = true ,
8429   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8430   rules .value_required:n = true ,
8431   code .tl_set:N = \l_@@_code_tl ,

```

```

8432     code .value_required:n = true ,
8433     unknown .code:n = \@_error:n { Unknown~key~for~SubMatrix }
8434 }

8435 \NewDocumentCommand \@_SubMatrix_in_code_before { m m m m ! 0 { } }
8436 {
8437     \peek_remove_spaces:n
8438     {
8439         \tl_gput_right:Ne \g_@@_pre_code_after_tl
8440         {
8441             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8442             [
8443                 delimiters / color = \l_@@_delimiters_color_tl ,
8444                 hlines = \l_@@_submatrix_hlines_clist ,
8445                 vlines = \l_@@_submatrix_vlines_clist ,
8446                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8447                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8448                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8449                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8450                 #5
8451             ]
8452         }
8453         \@_SubMatrix_in_code_before_i { #2 } { #3 }
8454     }
8455 }

8456 \NewDocumentCommand \@_SubMatrix_in_code_before_i
8457 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8458 { \@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8459 \cs_new_protected:Npn \@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8460 {
8461     \seq_gput_right:Ne \g_@@_submatrix_seq
8462     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8463     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8464     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8465     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8466     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8467 }
8468 }

```

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`.

```

8469 \hook_gput_code:nnn { begindocument } { . }
8470 {
8471     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8472     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8473 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8474 {
8475   \peek_remove_spaces:n
8476   {
8477     \@@_sub_matrix:nnnnnnn
8478     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8479   }
8480 }
8481 }

```

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).

```

8482 \NewDocumentCommand \@@_compute_i_j:nn
8483 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8484 { \@@_compute_i_j:nnnn #1 #2 }

8485 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8486 {
8487   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8488   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8489   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8490   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8491   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8492     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8493   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8494     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8495   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8496     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8497   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8498     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8499 }

8500 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8501 {
8502   \group_begin:

```

The four following token lists correspond to the position of the \backslash SubMatrix.

```

8503 \@@_compute_i_j:nn { #2 } { #3 }
8504 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8505   { \cs_set_nopar:Npn \arraystretch { 1 } }
8506 \bool_lazy_or:nnTF
8507   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8508   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8509   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8510 {
8511   \str_clear_new:N \l_@@_submatrix_name_str
8512   \keys_set:nn { nicematrix / SubMatrix } { #5 }
8513   \pgfpicture
8514   \pgfrememberpicturepositiononpagetrue
8515   \pgf@relevantforpicturesizefalse
8516   \pgfset { inner~sep = \c_zero_dim }
8517   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8518   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \backslash int_step_inline:nnn is provided by curriification.

```

8519 \bool_if:NTF \l_@@_submatrix_slim_bool
8520   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8521   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8522   {
8523     \cs_if_exist:cT
8524       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8525       {
8526         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8527         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim

```

```

8528         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8529     }
8530     \cs_if_exist:cT
8531     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8532     {
8533         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8534         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8535         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8536     }
8537 }
8538 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8539 { \@@_error:nn { Impossible~delimiter } { left } }
8540 {
8541     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8542     { \@@_error:nn { Impossible~delimiter } { right } }
8543     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8544 }
8545 \endpgfpicture
8546 }
8547 \group_end:
8548 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8549 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8550 {
8551     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8552     \dim_set:Nn \l_@@_y_initial_dim
8553     {
8554         \fp_to_dim:n
8555         {
8556             \pgf@y
8557             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8558         }
8559     }
8560     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8561     \dim_set:Nn \l_@@_y_final_dim
8562     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8563     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8564     {
8565         \cs_if_exist:cT
8566         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8567         {
8568             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8569             \dim_set:Nn \l_@@_y_initial_dim
8570             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8571         }
8572         \cs_if_exist:cT
8573         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8574         {
8575             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8576             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8577             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8578         }
8579     }
8580     \dim_set:Nn \l_tmpa_dim
8581     {
8582         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8583         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8584     }
8585     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8586 \group_begin:
8587 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8588 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8589 \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`.

```

8590 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8591 {
8592   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8593   {
8594     \int_compare:nNnT
8595       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8596     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8597       \@@_qpoint:n { col - ##1 }
8598       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8599       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8600       \pgfusepathqstroke
8601     }
8602   }
8603 }

```

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.

```

8604 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8605 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8606 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8607 {
8608   \bool_lazy_and:nnTF
8609   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8610   {
8611     \int_compare_p:nNn
8612     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8613   {
8614     \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8615     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8616     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8617     \pgfusepathqstroke
8618   }
8619   { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8620 }

```

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.

```

8621 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8622 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8623 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8624 {
8625   \bool_lazy_and:nnTF
8626   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8627   {
8628     \int_compare_p:nNn
8629     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8630   {
8631     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8632 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8633 \dim_set:Nn \l_tmpa_dim

```

```

8634         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8635     \str_case:nn { #1 }
8636     {
8637         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8638         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8639         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8640         }
8641     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8642     \dim_set:Nn \l_tmpb_dim
8643     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8644     \str_case:nn { #2 }
8645     {
8646         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8647         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8648         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8649     }
8650     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8651     \pgfusepathqstroke
8652     \group_end:
8653 }
8654 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8655 }

```

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).

```

8656     \str_if_empty:NF \l_@@_submatrix_name_str
8657     {
8658         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8659         \l_@@_x_initial_dim \l_@@_y_initial_dim
8660         \l_@@_x_final_dim \l_@@_y_final_dim
8661     }
8662     \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`.

```

8663     \begin { pgfscope }
8664     \pgftransformshift
8665     {
8666         \pgfpoint
8667         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8668         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8669     }
8670     \str_if_empty:NTF \l_@@_submatrix_name_str
8671     { \@@_node_left:nn #1 { } }
8672     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8673     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8674     \pgftransformshift
8675     {
8676         \pgfpoint
8677         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8678         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8679     }
8680     \str_if_empty:NTF \l_@@_submatrix_name_str
8681     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8682     {
8683         \@@_node_right:nnnn #2
8684         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8685     }

```

```

8686 \cs_set_eq:NN \pgfpointanchor \@_pgfpointanchor:n
8687 \flag_clear_new:N \l_@@_code_flag
8688 \l_@@_code_tl
8689 }

```

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`.

```

8690 \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.

```

8691 \cs_new_protected:Npn \@_pgfpointanchor:n #1
8692 {
8693   \use:e
8694   { \exp_not:N \@_old_pgfpointanchor { \@_pgfpointanchor_i:nn #1 } }
8695 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where "name_of_node" is the name of the Tikz node without the potential prefix and suffix. That's why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```

8696 \cs_new:Npn \@_pgfpointanchor_i:nn #1 #2
8697 { #1 { \@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8698 \tl_const:Nn \c_@@_integers_alist_tl
8699 {
8700   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8701   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8702   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8703   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8704 }

```

```

8705 \cs_new:Npn \@_pgfpointanchor_ii:w #1-#2\q_stop
8706 {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form $i-|j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

8707 \tl_if_empty:nTF { #2 }
8708 {
8709   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8710   {
8711     \flag_raise:N \l_@@_code_flag
8712     \int_if_even:nTF { \flag_height:N \l_@@_code_if_flag }
8713     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8714     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8715   }
8716   { #1 }
8717 }

```


If there is an hyphen, we have to see whether we have a node of the form $i-j$, $row-i$ or $col-j$.

```
8718     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8719 }
```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```
8720 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8721   {
8722     \str_case:nnF { #1 }
8723     {
8724       { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8725       { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8726     }

```

Now the case of a node of the form $i-j$.

```
8727     {
8728       \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8729       - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8730     }
8731 }
```

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`).

```
8732 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8733   {
8734     \pgfnode
8735     { rectangle }
8736     { east }
8737     {
8738       \nullfont
8739       \c_math_toggle_token
8740       \@@_color:o \l_@@_delimiters_color_tl
8741       \left #1
8742       \vcenter
8743       {
8744         \nullfont
8745         \hrule \@height \l_tmpa_dim
8746         \@depth \c_zero_dim
8747         \@width \c_zero_dim
8748       }
8749       \right .
8750       \c_math_toggle_token
8751     }
8752     { #2 }
8753     { }
8754   }
```

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.

```
8755 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8756   {
8757     \pgfnode
8758     { rectangle }
8759     { west }
8760     {
8761       \nullfont
8762       \c_math_toggle_token
8763       \colorlet { current-color } { . }
8764       \@@_color:o \l_@@_delimiters_color_tl
8765       \left .

```

```

8766     \vcenter
8767     {
8768         \nullfont
8769         \hrule \@height \l_tmpa_dim
8770             \@depth \c_zero_dim
8771             \@width \c_zero_dim
8772     }
8773     \right #1
8774     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8775     ~ { \color { current-color } \smash { #4 } }
8776     \c_math_toggle_token
8777 }
8778 { #2 }
8779 { }
8780 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8781 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8782 {
8783     \peek_remove_spaces:n
8784     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8785 }
8786 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8787 {
8788     \peek_remove_spaces:n
8789     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8790 }
8791 \keys_define:nn { nicematrix / Brace }
8792 {
8793     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8794     left-shorten .default:n = true ,
8795     left-shorten .value_forbidden:n = true ,
8796     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8797     right-shorten .default:n = true ,
8798     right-shorten .value_forbidden:n = true ,
8799     shorten .meta:n = { left-shorten , right-shorten } ,
8800     shorten .value_forbidden:n = true ,
8801     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8802     yshift .value_required:n = true ,
8803     yshift .initial:n = \c_zero_dim ,
8804     color .tl_set:N = \l_tmpa_tl ,
8805     color .value_required:n = true ,
8806     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8807 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#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`.

```

8808 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8809 {
8810     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8811 \@@_compute_i_j:nn { #1 } { #2 }
8812 \bool_lazy_or:nnTF

```

```

8813 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8814 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8815 {
8816   \str_if_eq:eeTF { #5 } { under }
8817   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8818   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8819 }
8820 {
8821   \tl_clear:N \l_tmpa_tl
8822   \keys_set:nn { nicematrix / Brace } { #4 }
8823   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8824   \pgfpicture
8825   \pgfrememberpicturepositiononpagetrue
8826   \pgf@relevantforpicturesizefalse
8827   \bool_if:NT \l_@@_brace_left_shorten_bool
8828   {
8829     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8830     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8831     {
8832       \cs_if_exist:cT
8833       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8834       {
8835         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8836
8837         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8838         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8839       }
8840     }
8841   }
8842   \bool_lazy_or:nnT
8843   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8844   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8845   {
8846     \@@_qpoint:n { col - \l_@@_first_j_tl }
8847     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8848   }
8849   \bool_if:NT \l_@@_brace_right_shorten_bool
8850   {
8851     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8852     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8853     {
8854       \cs_if_exist:cT
8855       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8856       {
8857         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8858         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8859         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8860       }
8861     }
8862   }
8863   \bool_lazy_or:nnT
8864   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8865   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8866   {
8867     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8868     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8869   }
8870   \pgfset { inner~sep = \c_zero_dim }
8871   \str_if_eq:eeTF { #5 } { under }
8872   { \@@_underbrace_i:n { #3 } }
8873   { \@@_overbrace_i:n { #3 } }
8874   \endpgfpicture
8875 }

```

```

8876   \group_end:
8877 }

```

The argument is the text to put above the brace.

```

8878 \cs_new_protected:Npn \@@_overbrace_i:n #1
8879 {
8880   \@@_qpoint:n { row - \l_@@_first_i_tl }
8881   \pgftransformshift
8882     {
8883     \pgfpoint
8884       { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8885       { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8886     }
8887   \pgfnode
8888     { rectangle }
8889     { south }
8890     {
8891     \vtop
8892       {
8893       \group_begin:
8894       \everycr { }
8895       \halign
8896         {
8897         \hfil ## \hfil \crcr
8898         \bool_if:NTF \l_@@_tabular_bool
8899           { \begin { tabular } { c } #1 \end { tabular } }
8900           { $ \begin { array } { c } #1 \end { array } $ }
8901         \cr
8902         \c_math_toggle_token
8903         \overbrace
8904           {
8905           \hbox_to_wd:nn
8906             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8907             { }
8908           }
8909         \c_math_toggle_token
8910         \cr
8911         }
8912       \group_end:
8913     }
8914   }
8915   { }
8916   { }
8917 }

```

The argument is the text to put under the brace.

```

8918 \cs_new_protected:Npn \@@_underbrace_i:n #1
8919 {
8920   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8921   \pgftransformshift
8922     {
8923     \pgfpoint
8924       { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8925       { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8926     }
8927   \pgfnode
8928     { rectangle }
8929     { north }
8930     {
8931     \group_begin:
8932     \everycr { }
8933     \vbox
8934     {

```

```

8935     \halign
8936     {
8937         \hfil ## \hfil \crcr
8938         \c_math_toggle_token
8939         \underbrace
8940         {
8941             \hbox_to_wd:nn
8942             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8943             { }
8944         }
8945         \c_math_toggle_token
8946         \cr
8947         \bool_if:NTF \l_@@_tabular_bool
8948         { \begin { tabular } { c } #1 \end { tabular } }
8949         { $ \begin { array } { c } #1 \end { array } $ }
8950         \cr
8951     }
8952 }
8953 \group_end:
8954 }
8955 { }
8956 { }
8957 }

```

35 The command TikzEveryCell

```

8958 \bool_new:N \l_@@_not_empty_bool
8959 \bool_new:N \l_@@_empty_bool
8960
8961 \keys_define:nn { nicematrix / TikzEveryCell }
8962 {
8963     not-empty .code:n =
8964         \bool_lazy_or:nnTF
8965         \l_@@_in_code_after_bool
8966         \g_@@_recreate_cell_nodes_bool
8967         { \bool_set_true:N \l_@@_not_empty_bool }
8968         { \@@_error:n { detection~of~empty~cells } } } ,
8969     not-empty .value_forbidden:n = true ,
8970     empty .code:n =
8971         \bool_lazy_or:nnTF
8972         \l_@@_in_code_after_bool
8973         \g_@@_recreate_cell_nodes_bool
8974         { \bool_set_true:N \l_@@_empty_bool }
8975         { \@@_error:n { detection~of~empty~cells } } } ,
8976     empty .value_forbidden:n = true ,
8977     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8978 }
8979
8980
8981 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8982 {
8983     \IfPackageLoadedTF { tikz }
8984     {
8985         \group_begin:
8986         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

8987     \tl_set:Nn \l_tmpa_tl { { #2 } }
8988     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

8989         { \@@_for_a_block:nnnnn ##1 }
8990     \@@_all_the_cells:
8991     \group_end:
8992 }
8993 { \@@_error:n { TikzEveryCell~without~tikz } }
8994 }
8995
8996 \tl_new:N \@@_i_tl
8997 \tl_new:N \@@_j_tl
8998
8999
9000 \cs_new_protected:Nn \@@_all_the_cells:
9001 {
9002     \int_step_variable:nNn \c@iRow \@@_i_tl
9003     {
9004         \int_step_variable:nNn \c@jCol \@@_j_tl
9005         {
9006             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9007             {
9008                 \clist_if_in:NcF \l_@@_corners_cells_clist
9009                 { \@@_i_tl - \@@_j_tl }
9010                 {
9011                     \bool_set_false:N \l_tmpa_bool
9012                     \cs_if_exist:cTF
9013                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9014                     {
9015                         \bool_if:NF \l_@@_empty_bool
9016                         { \bool_set_true:N \l_tmpa_bool }
9017                     }
9018                     {
9019                         \bool_if:NF \l_@@_not_empty_bool
9020                         { \bool_set_true:N \l_tmpa_bool }
9021                     }
9022                     \bool_if:NT \l_tmpa_bool
9023                     {
9024                         \@@_block_tikz:onnnn
9025                         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9026                     }
9027                 }
9028             }
9029         }
9030     }
9031 }
9032
9033 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9034 {
9035     \bool_if:NF \l_@@_empty_bool
9036     {
9037         \@@_block_tikz:onnnn
9038         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9039     }
9040     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9041 }
9042
9043 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9044 {
9045     \int_step_inline:nnn { #1 } { #3 }
9046     {
9047         \int_step_inline:nnn { #2 } { #4 }
9048         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9049     }
9050 }

```

36 The command \ShowCellNames

```

9051 \NewDocumentCommand \@@_ShowCellNames { }
9052 {
9053   \bool_if:NT \l_@@_in_code_after_bool
9054   {
9055     \pgfpicture
9056     \pgfrememberpicturepositiononpagetrue
9057     \pgf@relevantforpicturesizefalse
9058     \pgfpathrectanglecorners
9059     { \@@_qpoint:n { 1 } }
9060     {
9061       \@@_qpoint:n
9062       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9063     }
9064     \pgfsetfillopacity { 0.75 }
9065     \pgfsetfillcolor { white }
9066     \pgfusepathqfill
9067     \endpgfpicture
9068   }
9069   \dim_gzero_new:N \g_@@_tmpc_dim
9070   \dim_gzero_new:N \g_@@_tmpd_dim
9071   \dim_gzero_new:N \g_@@_tmpe_dim
9072   \int_step_inline:nn \c@iRow
9073   {
9074     \bool_if:NTF \l_@@_in_code_after_bool
9075     {
9076       \pgfpicture
9077       \pgfrememberpicturepositiononpagetrue
9078       \pgf@relevantforpicturesizefalse
9079     }
9080     { \begin { pgfpicture } }
9081     \@@_qpoint:n { row - ##1 }
9082     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9083     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9084     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9085     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9086     \bool_if:NTF \l_@@_in_code_after_bool
9087     { \endpgfpicture }
9088     { \end { pgfpicture } }
9089     \int_step_inline:nn \c@jCol
9090     {
9091       \hbox_set:Nn \l_tmpa_box
9092       {
9093         \normalfont \Large \sffamily \bfseries
9094         \bool_if:NTF \l_@@_in_code_after_bool
9095         { \color { red } }
9096         { \color { red ! 50 } }
9097         ##1 - #####1
9098       }
9099       \bool_if:NTF \l_@@_in_code_after_bool
9100       {
9101         \pgfpicture
9102         \pgfrememberpicturepositiononpagetrue
9103         \pgf@relevantforpicturesizefalse
9104       }
9105       { \begin { pgfpicture } }
9106       \@@_qpoint:n { col - #####1 }
9107       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9108       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9109       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9110       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9111     \bool_if:NTF \l_@@_in_code_after_bool
9112         { \endpgfpicture }
9113         { \end { pgfpicture } }
9114     \fp_set:Nn \l_tmpa_fp
9115         {
9116         \fp_min:nn
9117         {
9118         \fp_min:nn
9119             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9120             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9121         }
9122         { 1.0 }
9123     }
9124     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9125     \pgfpicture
9126     \pgfrememberpicturepositiononpagetrue
9127     \pgf@relevantforpicturesizefalse
9128     \pgftransformshift
9129     {
9130         \pgfpoint
9131         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9132         { \dim_use:N \g_tmpa_dim }
9133     }
9134     \pgfnode
9135     { rectangle }
9136     { center }
9137     { \box_use:N \l_tmpa_box }
9138     { }
9139     { }
9140     \endpgfpicture
9141 }
9142 }
9143 }

```

37 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.

```

9144 \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.

```

9145 \bool_new:N \g_@@_footnote_bool
9146 \msg_new:nmmm { nicematrix } { Unknown~key~for~package }
9147 {
9148     The~key~'\l_keys_key_str'~is~unknown. \\
9149     That~key~will~be~ignored. \\
9150     For~a~list~of~the~available~keys,~type~H~<return>.
9151 }
9152 {
9153     The~available~keys~are~(in~alphabetic~order):~
9154     footnote,~
9155     footnotehyper,~
9156     messages~for~Overleaf,~
9157     renew~dots,~and~
9158     renew~matrix.

```



```

9159 }
9160 \keys_define:nn { nicematrix / Package }
9161 {
9162   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9163   renew-dots .value_forbidden:n = true ,
9164   renew-matrix .code:n = \@@_renew_matrix: ,
9165   renew-matrix .value_forbidden:n = true ,
9166   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9167   footnote .bool_set:N = \g_@@_footnote_bool ,
9168   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. You keep the following key only for compatibility but maybe we will delete it.

```

9169   no-test-for-array .code:n = \prg_do_nothing: ,
9170   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9171 }
9172 \ProcessKeysOptions { nicematrix / Package }

```

```

9173 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9174 {
9175   You-can't-use-the-option-'footnote'~because~the~package~
9176   footnotehyper~has~already~been~loaded.~
9177   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9178   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9179   of~the~package~footnotehyper.\\
9180   The~package~footnote~won't~be~loaded.
9181 }

```

```

9182 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9183 {
9184   You-can't-use-the-option-'footnotehyper'~because~the~package~
9185   footnote~has~already~been~loaded.~
9186   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9187   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9188   of~the~package~footnote.\\
9189   The~package~footnotehyper~won't~be~loaded.
9190 }

```

```

9191 \bool_if:NT \g_@@_footnote_bool
9192 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9193   \IfClassLoadedTF { beamer }
9194     { \bool_set_false:N \g_@@_footnote_bool }
9195     {
9196       \IfPackageLoadedTF { footnotehyper }
9197         { \@@_error:n { footnote~with~footnotehyper~package } }
9198         { \usepackage { footnote } }
9199     }
9200 }

```

```

9201 \bool_if:NT \g_@@_footnotehyper_bool
9202 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9203   \IfClassLoadedTF { beamer }
9204     { \bool_set_false:N \g_@@_footnote_bool }
9205     {
9206       \IfPackageLoadedTF { footnote }
9207         { \@@_error:n { footnotehyper~with~footnote~package } }
9208         { \usepackage { footnotehyper } }
9209     }

```

```

9210   \bool_set_true:N \g_@@_footnote_bool
9211 }

```

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}`.

38 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.

```

9212 \bool_new:N \l_@@_underscore_loaded_bool
9213 \IfPackageLoadedT { underscore }
9214 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9215 \hook_gput_code:nnn { begindocument } { . }
9216 {
9217   \bool_if:NF \l_@@_underscore_loaded_bool
9218   {
9219     \IfPackageLoadedT { underscore }
9220     { \@@_error:n { underscore-after~nicematrix } }
9221   }
9222 }

```

39 Error messages of the package

```

9223 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9224 { \str_const:Nn \c_@@_available_keys_str { } }
9225 {
9226   \str_const:Nn \c_@@_available_keys_str
9227   { For~a~list~of~the~available~keys,~type-H~<return>. }
9228 }
9229 \seq_new:N \g_@@_types_of_matrix_seq
9230 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9231 {
9232   NiceMatrix ,
9233   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9234 }
9235 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9236 { \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`.

```

9237 \cs_new_protected:Npn \@@_error_too_much_cols:
9238 {
9239   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9240   { \@@_fatal:nm { too-much-cols-for-array } }
9241   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9242   { \@@_fatal:n { too-much-cols-for-matrix } }
9243   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9244   { \@@_fatal:n { too-much-cols-for-matrix } }
9245   \bool_if:NF \l_@@_last_col_without_value_bool
9246   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }

```

9247 }

The following command must *not* be protected since it's used in an error message.

```
9248 \cs_new:Npn \@@_message_hdotsfor:
9249 {
9250   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9251     { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
9252 }
9253 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9254 {
9255   Incompatible-options.\\
9256   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at-this-time.\\
9257   The-output-will-not-be-reliable.
9258 }
9259 \@@_msg_new:nn { negative-weight }
9260 {
9261   Negative-weight.\\
9262   The-weight-of-the-'X'-columns-must-be-positive-and-you-have-used-
9263   the-value~'\int_use:N \l_@@_weight_int'.\\
9264   The-absolute-value-will-be-used.
9265 }
9266 \@@_msg_new:nn { last-col-not-used }
9267 {
9268   Column-not-used.\\
9269   The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column-
9270   in-your~\@@_full_name_env:~.~However,~you-can-go-on.
9271 }
9272 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9273 {
9274   Too-much-columns.\\
9275   In-the-row~\int_eval:n { \c@iRow },~
9276   you-try-to-use-more-columns-
9277   than-allowed-by-your~\@@_full_name_env:~.\@@_message_hdotsfor:\
9278   The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9279   (plus-the-exterior-columns).~This-error-is-fatal.
9280 }
9281 \@@_msg_new:nn { too-much-cols-for-matrix }
9282 {
9283   Too-much-columns.\\
9284   In-the-row~\int_eval:n { \c@iRow },~
9285   you-try-to-use-more-columns-than-allowed-by-your~
9286   \@@_full_name_env:~.\@@_message_hdotsfor:\ Recall~that~the-maximal-
9287   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9288   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9289   Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9290   \token_to_str:N \setcounter\ to~change~that~value).~
9291   This-error-is-fatal.
9292 }
9293 \@@_msg_new:nn { too-much-cols-for-array }
9294 {
9295   Too-much-columns.\\
9296   In-the-row~\int_eval:n { \c@iRow },~
9297   ~you-try-to-use-more-columns~than~allowed-by~your~
9298   \@@_full_name_env:~.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9299   \int_use:N \g_@@_static_num_of_col_int\
9300   ~ (plus~the~potential~exterior~ones).~
9301   This-error-is-fatal.
9302 }
9303 \@@_msg_new:nn { columns-not-used }
9304 {
9305   Columns-not-used.\\
```

```

9306     The-preamble-of-your-@@_full_name_env:\ announces-\int_use:N
9307     \g_@@_static_num_of_col_int\ columns-but-you-use-only-\int_use:N \c@jCol.\\
9308     The-columns-you-did-not-used-won't-be-created.\\
9309     You-won't-have-similar-error-message-till-the-end-of-the-document.
9310 }
9311 \@@_msg_new:nn { empty-preamble }
9312 {
9313     Empty-preamble.\\
9314     The-preamble-of-your-@@_full_name_env:\ is-empty.\\
9315     This-error-is-fatal.
9316 }
9317 \@@_msg_new:nn { in-first-col }
9318 {
9319     Erroneous-use.\\
9320     You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9321     That-command-will-be-ignored.
9322 }
9323 \@@_msg_new:nn { in-last-col }
9324 {
9325     Erroneous-use.\\
9326     You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9327     That-command-will-be-ignored.
9328 }
9329 \@@_msg_new:nn { in-first-row }
9330 {
9331     Erroneous-use.\\
9332     You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9333     That-command-will-be-ignored.
9334 }
9335 \@@_msg_new:nn { in-last-row }
9336 {
9337     You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9338     That-command-will-be-ignored.
9339 }
9340 \@@_msg_new:nn { caption-outside-float }
9341 {
9342     Key-caption-forbidden.\\
9343     You-can't-use-the-key-'caption'-because-you-are-not-in-a-floating-
9344     environment.~This-key-will-be-ignored.
9345 }
9346 \@@_msg_new:nn { short-caption-without-caption }
9347 {
9348     You-should-not-use-the-key-'short-caption'~without-'caption'.~
9349     However,-your-'short-caption'~will-be-used-as-'caption'.
9350 }
9351 \@@_msg_new:nn { double-closing-delimiter }
9352 {
9353     Double-delimiter.\\
9354     You-can't-put-a-second-closing-delimiter-"#1"~just-after-a-first-closing-
9355     delimiter.~This-delimiter-will-be-ignored.
9356 }
9357 \@@_msg_new:nn { delimiter-after-opening }
9358 {
9359     Double-delimiter.\\
9360     You-can't-put-a-second-delimiter-"#1"~just-after-a-first-opening-
9361     delimiter.~That-delimiter-will-be-ignored.
9362 }
9363 \@@_msg_new:nn { bad-option-for-line-style }
9364 {

```

```

9365     Bad-line-style.\
9366     Since-you-haven't-loaded-Tikz,~the-only~value~you~can~give~to~'line-style'~
9367     is~'standard'.~That~key~will~be~ignored.
9368 }
9369 \@_msg_new:nn { Identical-notes-in-caption }
9370 {
9371     Identical~tabular~notes.\
9372     You~can't~put~several~notes~with~the~same~content~in~
9373     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
9374     If~you~go~on,~the~output~will~probably~be~erroneous.
9375 }
9376 \@_msg_new:nn { tabularnote~below~the~tabular }
9377 {
9378     \token_to_str:N \tabularnote\ forbidden\
9379     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9380     of~your~tabular~because~the~caption~will~be~composed~below~
9381     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9382     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\
9383     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9384     no~similar~error~will~raised~in~this~document.
9385 }
9386 \@_msg_new:nn { Unknown~key~for~rules }
9387 {
9388     Unknown~key.\
9389     There~is~only~two~keys~available~here:~width~and~color.\
9390     Your~key~'\l_keys_key_str'~will~be~ignored.
9391 }
9392 \@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9393 {
9394     Unknown~key.\
9395     There~is~only~two~keys~available~here:~
9396     'empty'~and~'not~empty'.\
9397     Your~key~'\l_keys_key_str'~will~be~ignored.
9398 }
9399 \@_msg_new:nn { Unknown~key~for~rotate }
9400 {
9401     Unknown~key.\
9402     The~only~key~available~here~is~'c'.\
9403     Your~key~'\l_keys_key_str'~will~be~ignored.
9404 }
9405 \@_msg_new:nnn { Unknown~key~for~custom~line }
9406 {
9407     Unknown~key.\
9408     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9409     It~you~go~on,~you~will~probably~have~other~errors. \
9410     \c_@@_available_keys_str
9411 }
9412 {
9413     The~available~keys~are~(in~alphabetic~order):~
9414     ccommand,~
9415     color,~
9416     command,~
9417     dotted,~
9418     letter,~
9419     multiplicity,~
9420     sep~color,~
9421     tikz,~and~total~width.
9422 }
9423 \@_msg_new:nnn { Unknown~key~for~xdots }
9424 {
9425     Unknown~key.\

```

```

9426     The-key~'\l_keys_key_str'~is-unknown~for~a~command~for~drawing~dotted~rules.\\
9427     \c_@@_available_keys_str
9428   }
9429   {
9430     The-available-keys-are-(in-alphabetic-order):~
9431     'color',~
9432     'horizontal-labels',~
9433     'inter',~
9434     'line-style',~
9435     'radius',~
9436     'shorten',~
9437     'shorten-end'~and~'shorten-start'.
9438   }

9439 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9440 {
9441   Unknown-key.\\
9442   As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
9443   (and-you-try-to-use~'\l_keys_key_str')\\
9444   That-key-will-be-ignored.
9445 }

9446 \@@_msg_new:nn { label-without-caption }
9447 {
9448   You-can't-use-the-key~'label'~in-your~'{NiceTabular}'~because~
9449   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9450 }

9451 \@@_msg_new:nn { W-warning }
9452 {
9453   Line~\msg_line_number:~The-cell-is-too-wide-for-your-column~'W'~
9454   (row~\int_use:N \c@iRow).
9455 }

9456 \@@_msg_new:nn { Construct-too-large }
9457 {
9458   Construct-too-large.\\
9459   Your~command~\token_to_str:N #1
9460   can't-be-drawn-because-your-matrix-is-too-small.\\
9461   That-command-will-be-ignored.
9462 }

9463 \@@_msg_new:nn { underscore-after-nicematrix }
9464 {
9465   Problem-with-'underscore'.\\
9466   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9467   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9468   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9469 }

9470 \@@_msg_new:nn { ampersand-in-light-syntax }
9471 {
9472   Ampersand-forbidden.\\
9473   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9474   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9475 }

9476 \@@_msg_new:nn { double-backslash-in-light-syntax }
9477 {
9478   Double-backslash-forbidden.\\
9479   You-can't-use~\token_to_str:N
9480   \\~to-separate-rows-because-the-key~'light-syntax'~
9481   is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9482   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9483 }

9484 \@@_msg_new:nn { hlines-with-color }
9485 {

```

```

9486     Incompatible-keys.\
9487     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for~a~
9488     '\token_to_str:N \Block'~when~the-key~'color'~or~'draw'~is~used.\
9489     However,~you-can~put~several~commands~\token_to_str:N \Block.\
9490     Your-key-will-be-discarded.
9491 }

9492 \@@_msg_new:nn { bad-value-for-baseline }
9493 {
9494     Bad-value-for-baseline.\
9495     The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9496     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9497     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9498     the-form-'line-i'.\
9499     A-value-of~1~will-be-used.
9500 }

9501 \@@_msg_new:nn { detection-of-empty-cells }
9502 {
9503     Problem-with-'not-empty'\
9504     For-technical-reasons,~you-must-activate~
9505     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9506     in-order-to-use-the-key~'\l_keys_key_str'.\
9507     That-key-will-be-ignored.
9508 }

9509 \@@_msg_new:nn { siunitx-not-loaded }
9510 {
9511     siunitx-not-loaded\
9512     You-can't-use-the-columns-'S'~because-'siunitx'~is~not~loaded.\
9513     That-error-is-fatal.
9514 }

9515 \@@_msg_new:nn { Invalid-name }
9516 {
9517     Invalid-name.\
9518     You-can't-give-the-name~'\l_keys_value_tl'~to~a~\token_to_str:N
9519     \SubMatrix\ of~your~\@@_full_name_env:.\
9520     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
9521     This-key-will-be-ignored.
9522 }

9523 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9524 {
9525     Wrong-line.\
9526     You-try-to-draw~a~#1~line-of-number~'#2'~in~a~
9527     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9528     number~is~not~valid.~It~will-be-ignored.
9529 }

9530 \@@_msg_new:nn { Impossible-delimiter }
9531 {
9532     Impossible-delimiter.\
9533     It's-impossible-to-draw~the~#1~delimiter~of~your~
9534     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9535     in~that~column.
9536     \bool_if:NT \l_@@_submatrix_slim_bool
9537     { ~Maybe-you-should-try-without-the-key-'slim'. } \
9538     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9539 }

9540 \@@_msg_new:nnn { width-without-X-columns }
9541 {
9542     You-have-used-the-key~'width'~but~you-have-put~no~'X'~column.~
9543     That-key-will-be-ignored.
9544 }
9545 {
9546     This-message-is~the-message-'width-without-X-columns'~

```

```

9547   of~the~module~'nicematrix'.~
9548   The~experimented~users~can~disable~that~message~with~
9549   \token_to_str:N \msg_redirect_name:nnn.\\
9550 }
9551
9552 \@@_msg_new:nn { key~multiplicity~with~dotted }
9553 {
9554   Incompatible~keys. \\
9555   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9556   in~a~'custom~line'.~They~are~incompatible. \\
9557   The~key~'multiplicity'~will~be~discarded.
9558 }
9559 \@@_msg_new:nn { empty~environment }
9560 {
9561   Empty~environment.\\
9562   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9563 }
9564 \@@_msg_new:nn { No~letter~and~no~command }
9565 {
9566   Erroneous~use.\\
9567   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
9568   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9569   '~command'~(to~draw~horizontal~rules).\\
9570   However,~you~can~go~on.
9571 }
9572 \@@_msg_new:nn { Forbidden~letter }
9573 {
9574   Forbidden~letter.\\
9575   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9576   It~will~be~ignored.
9577 }
9578 \@@_msg_new:nn { Several~letters }
9579 {
9580   Wrong~name.\\
9581   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9582   have~used~'\l_@@_letter_str').\\
9583   It~will~be~ignored.
9584 }
9585 \@@_msg_new:nn { Delimiter~with~small }
9586 {
9587   Delimiter~forbidden.\\
9588   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9589   because~the~key~'small'~is~in~force.\\
9590   This~error~is~fatal.
9591 }
9592 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9593 {
9594   Unknown~cell.\\
9595   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9596   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9597   can't~be~executed~because~a~cell~doesn't~exist.\\
9598   This~command~\token_to_str:N \line\ will~be~ignored.
9599 }
9600 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9601 {
9602   Duplicate~name.\\
9603   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9604   in~this~\@@_full_name_env:.\\
9605   This~key~will~be~ignored.\\
9606   \bool_if:NF \g_@@_messages_for_Overleaf_bool

```



```

9607     { For-a-list-of-the-names-already-used,-type-H<return>. }
9608   }
9609   {
9610     The-names-already-defined-in-this-\@@_full_name_env:\ are:~
9611     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9612   }
9613 \@@_msg_new:nn { r~or~l~with~preamble }
9614 {
9615   Erroneous-use.\\
9616   You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~.~
9617   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
9618   your~\@@_full_name_env:~.\\
9619   This-key-will-be-ignored.
9620 }
9621 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9622 {
9623   Erroneous-use.\\
9624   You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9625   the~array.~This~error~is~fatal.
9626 }
9627 \@@_msg_new:nn { bad~corner }
9628 {
9629   Bad-corner.\\
9630   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9631   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9632   This~specification~of~corner~will~be~ignored.
9633 }
9634 \@@_msg_new:nn { bad~border }
9635 {
9636   Bad-border.\\
9637   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9638   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9639   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9640   also~use~the~key~'tikz'
9641   \IfPackageLoadedF { tikz }
9642   {-if~you~load~the~LaTeX~package~'tikz'}).\\
9643   This~specification~of~border~will~be~ignored.
9644 }
9645 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9646 {
9647   TikZ~not~loaded.\\
9648   You-can't-use~\token_to_str:N \TikzEveryCell\
9649   because~you~have~not~loaded~tikz.~
9650   This~command~will~be~ignored.
9651 }
9652 \@@_msg_new:nn { tikz~key~without~tikz }
9653 {
9654   TikZ~not~loaded.\\
9655   You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
9656   \Block'~because~you~have~not~loaded~tikz.~
9657   This~key~will~be~ignored.
9658 }
9659 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9660 {
9661   Erroneous-use.\\
9662   In~the~\@@_full_name_env:,~you~must~use~the~key~
9663   'last~col'~without~value.\\
9664   However,~you~can~go~on~for~this~time~
9665   (the~value~'\l_keys_value_tl'~will~be~ignored).
9666 }

```

```

9667 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9668 {
9669   Erroneous-use.\
9670   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9671   'last-col'~without-value.\
9672   However,~you-can-go-on-for-this-time~
9673   (the-value~'\l_keys_value_tl'~will-be-ignored).
9674 }

9675 \@@_msg_new:nn { Block-too-large-1 }
9676 {
9677   Block-too-large.\
9678   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
9679   too-small-for-that-block. \
9680   This-block-and-maybe-others-will-be-ignored.
9681 }

9682 \@@_msg_new:nn { Block-too-large-2 }
9683 {
9684   Block-too-large.\
9685   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9686   \g_@@_static_num_of_col_int\
9687   columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9688   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9689   (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9690   This-block-and-maybe-others-will-be-ignored.
9691 }

9692 \@@_msg_new:nn { unknown-column-type }
9693 {
9694   Bad-column-type.\
9695   The-column-type~'#1'~in-your~\@@_full_name_env:\
9696   is-unknown. \
9697   This-error-is-fatal.
9698 }

9699 \@@_msg_new:nn { unknown-column-type-S }
9700 {
9701   Bad-column-type.\
9702   The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \
9703   If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
9704   load-that-package. \
9705   This-error-is-fatal.
9706 }

9707 \@@_msg_new:nn { tabularnote-forbidden }
9708 {
9709   Forbidden-command.\
9710   You-can't-use-the-command~\token_to_str:N\tabularnote\
9711   ~here.~This-command-is-available-only-in~
9712   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9713   the-argument-of-a-command~\token_to_str:N \caption\ included~
9714   in-an-environment~{table}. \
9715   This-command-will-be-ignored.
9716 }

9717 \@@_msg_new:nn { borders-forbidden }
9718 {
9719   Forbidden-key.\
9720   You-can't-use-the-key~'borders'~of~the-command~\token_to_str:N \Block\
9721   because-the-option~'rounded-corners'~
9722   is-in-force-with-a-non-zero-value.\
9723   This-key-will-be-ignored.
9724 }

9725 \@@_msg_new:nn { bottomrule-without-booktabs }
9726 {
9727   booktabs-not-loaded.\

```

```

9728     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9729     loaded~'booktabs'.\\
9730     This~key~will~be~ignored.
9731 }

9732 \@@_msg_new:nn { enumitem~not~loaded }
9733 {
9734     enumitem~not~loaded.\\
9735     You~can't~use~the~command~\token_to_str:N\tabularnote\
9736     ~because~you~haven't~loaded~'enumitem'.\\
9737     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9738     ignored~in~the~document.
9739 }

9740 \@@_msg_new:nn { tikz~without~tikz }
9741 {
9742     Tikz~not~loaded.\\
9743     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9744     loaded.~If~you~go~on,~that~key~will~be~ignored.
9745 }

9746 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9747 {
9748     Tikz~not~loaded.\\
9749     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9750     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9751     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9752     use~that~custom~line.
9753 }

9754 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9755 {
9756     Tikz~not~loaded.\\
9757     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9758     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9759     That~key~will~be~ignored.
9760 }

9761 \@@_msg_new:nn { without~color~inside }
9762 {
9763     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9764     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9765     outside~\token_to_str:N \CodeBefore,~you~
9766     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9767     You~can~go~on~but~you~may~need~more~compilations.
9768 }

9769 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9770 {
9771     Erroneous~use.\\
9772     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9773     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9774     The~key~'color'~will~be~discarded.
9775 }

9776 \@@_msg_new:nn { Wrong~last~row }
9777 {
9778     Wrong~number.\\
9779     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9780     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9781     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9782     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9783     without~value~(more~compilations~might~be~necessary).
9784 }

9785 \@@_msg_new:nn { Yet~in~env }
9786 {
9787     Nested~environments.\\

```

```

9788     Environments-of-nicematrix-can't-be-nested.\\
9789     This-error-is-fatal.
9790 }
9791 \@@_msg_new:nn { Outside-math-mode }
9792 {
9793     Outside-math-mode.\\
9794     The-\@@_full_name_env:\ can-be-used-only-in-math-mode~
9795     (and-not-in-\token_to_str:N \vcenter).\\
9796     This-error-is-fatal.
9797 }
9798 \@@_msg_new:nn { One-letter-allowed }
9799 {
9800     Bad-name.\\
9801     The-value-of-key-\l_keys_key_str'~must-be-of-length-1.\\
9802     It-will-be-ignored.
9803 }
9804 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9805 {
9806     Environment-{TabularNote}-forbidden.\\
9807     You-must-use-{TabularNote}-at-the-end-of-your-{NiceTabular}-
9808     but-*before*-the-\token_to_str:N \CodeAfter.\\
9809     This-environment-{TabularNote}-will-be-ignored.
9810 }
9811 \@@_msg_new:nn { varwidth-not-loaded }
9812 {
9813     varwidth-not-loaded.\\
9814     You-can't-use-the-column-type-'V'~because-'varwidth'~is-not~
9815     loaded.\\
9816     Your-column-will-behave-like-'p'.
9817 }
9818 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9819 {
9820     Unknow-key.\\
9821     Your-key-\l_keys_key_str'~is-unknown-for-a-rule.\\
9822     \c_@@_available_keys_str
9823 }
9824 {
9825     The-available-keys-are-(in-alphabetic-order):~
9826     color,~
9827     dotted,~
9828     multiplicity,~
9829     sep-color,~
9830     tikz,~and~total-width.
9831 }
9832
9833 \@@_msg_new:nnn { Unknow-key-for-Block }
9834 {
9835     Unknow-key.\\
9836     The-key-\l_keys_key_str'~is-unknown-for-the-command-\token_to_str:N
9837     \Block.\\ It-will-be-ignored. \\
9838     \c_@@_available_keys_str
9839 }
9840 {
9841     The-available-keys-are-(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
9842     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9843     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9844     and~vlines.
9845 }
9846 \@@_msg_new:nnn { Unknow-key-for-Brace }
9847 {
9848     Unknow-key.\\

```

```

9849 The-key~'\l_keys_key_str'~is-unknown~for~the~commands~\token_to_str:N
9850 \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9851 It~will~be~ignored. \\
9852 \c_@@_available_keys_str
9853 }
9854 {
9855 The-available-keys-are-(in-alphabetic-order):~color,~left-shorten,~
9856 right-shorten,~shorten-(which-fixes-both-left-shorten-and-
9857 right-shorten)~and~yshift.
9858 }
9859 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9860 {
9861 Unknown~key.\\
9862 The-key~'\l_keys_key_str'~is-unknown.\\
9863 It~will~be~ignored. \\
9864 \c_@@_available_keys_str
9865 }
9866 {
9867 The-available-keys-are-(in-alphabetic-order):~
9868 delimiters/color,~
9869 rules~(with~the~subkeys~'color'~and~'width'),~
9870 sub-matrix~(several~subkeys)~
9871 and~xdots~(several~subkeys).~
9872 The-latter-is-for-the-command~\token_to_str:N \line.
9873 }
9874 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9875 {
9876 Unknown~key.\\
9877 The-key~'\l_keys_key_str'~is-unknown.\\
9878 It~will~be~ignored. \\
9879 \c_@@_available_keys_str
9880 }
9881 {
9882 The-available-keys-are-(in-alphabetic-order):~
9883 create-cell-nodes,~
9884 delimiters/color~and~
9885 sub-matrix~(several~subkeys).
9886 }
9887 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9888 {
9889 Unknown~key.\\
9890 The-key~'\l_keys_key_str'~is-unknown.\\
9891 That-key-will-be-ignored. \\
9892 \c_@@_available_keys_str
9893 }
9894 {
9895 The-available-keys-are-(in-alphabetic-order):~
9896 'delimiters/color',~
9897 'extra-height',~
9898 'hlines',~
9899 'hvlines',~
9900 'left-xshift',~
9901 'name',~
9902 'right-xshift',~
9903 'rules'~(with~the~subkeys~'color'~and~'width'),~
9904 'slim',~
9905 'vlines'~and~'xshift'~(which-sets~both~'left-xshift'~
9906 and~'right-xshift').\\
9907 }
9908 \@@_msg_new:nnn { Unknown~key~for~notes }
9909 {
9910 Unknown~key.\\

```

```

9911     The-key~'\l_keys_key_str'~is-unknown.\\
9912     That-key-will-be-ignored. \\
9913     \c_@@_available_keys_str
9914 }
9915 {
9916     The-available-keys-are-(in~alphabetic~order):~
9917     bottomrule,~
9918     code-after,~
9919     code-before,~
9920     detect-duplicates,~
9921     enumitem-keys,~
9922     enumitem-keys-para,~
9923     para,~
9924     label-in-list,~
9925     label-in-tabular~and~
9926     style.
9927 }

9928 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9929 {
9930     Unknown~key.\\
9931     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9932     \token_to_str:N \RowStyle. \\
9933     That-key-will-be-ignored. \\
9934     \c_@@_available_keys_str
9935 }
9936 {
9937     The-available-keys-are-(in~alphabetic~order):~
9938     'bold',~
9939     'cell-space-top-limit',~
9940     'cell-space-bottom-limit',~
9941     'cell-space-limits',~
9942     'color',~
9943     'nb-rows'~and~
9944     'rowcolor'.
9945 }

9946 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9947 {
9948     Unknown~key.\\
9949     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9950     \token_to_str:N \NiceMatrixOptions. \\
9951     That-key-will-be-ignored. \\
9952     \c_@@_available_keys_str
9953 }
9954 {
9955     The-available-keys-are-(in~alphabetic~order):~
9956     &~in~blocks,~
9957     allow-duplicate-names,~
9958     ampersand-in-blocks,~
9959     caption-above,~
9960     cell-space-bottom-limit,~
9961     cell-space-limits,~
9962     cell-space-top-limit,~
9963     code-for-first-col,~
9964     code-for-first-row,~
9965     code-for-last-col,~
9966     code-for-last-row,~
9967     corners,~
9968     custom-key,~
9969     create-extra-nodes,~
9970     create-medium-nodes,~
9971     create-large-nodes,~
9972     custom-line,~
9973     delimiters~(several~subkeys),~

```

```

9974   end-of-row,~
9975   first-col,~
9976   first-row,~
9977   hlines,~
9978   hvlines,~
9979   hvlines-except-borders,~
9980   last-col,~
9981   last-row,~
9982   left-margin,~
9983   light-syntax,~
9984   light-syntax-expanded,~
9985   matrix/columns-type,~
9986   no-cell-nodes,~
9987   notes~(several~subkeys),~
9988   nullify-dots,~
9989   pgf-node-code,~
9990   renew-dots,~
9991   renew-matrix,~
9992   respect-arraystretch,~
9993   rounded-corners,~
9994   right-margin,~
9995   rules~(with~the~subkeys~'color'~and~'width'),~
9996   small,~
9997   sub-matrix~(several~subkeys),~
9998   vlins,~
9999   xdots~(several~subkeys).
10000 }

```

For `{NiceArray}`, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

10001 \@@_msg_new:nnn { Unknown-key-for-NiceArray }
10002 {
10003   Unknown-key.\\
10004   The-key~'\l_keys_key_str'~is-unknown-for-the-environment-
10005   \{NiceArray\}. \\
10006   That-key-will-be-ignored. \\
10007   \c_@@_available_keys_str
10008 }
10009 {
10010   The-available-keys-are-(in-alphabetic-order):~
10011   &-in-blocks,~
10012   ampersand-in-blocks,~
10013   b,~
10014   baseline,~
10015   c,~
10016   cell-space-bottom-limit,~
10017   cell-space-limits,~
10018   cell-space-top-limit,~
10019   code-after,~
10020   code-for-first-col,~
10021   code-for-first-row,~
10022   code-for-last-col,~
10023   code-for-last-row,~
10024   color-inside,~
10025   columns-width,~
10026   corners,~
10027   create-extra-nodes,~
10028   create-medium-nodes,~
10029   create-large-nodes,~
10030   extra-left-margin,~
10031   extra-right-margin,~
10032   first-col,~
10033   first-row,~
10034   hlines,~

```

```

10035     hvlines,~
10036     hvlines-except-borders,~
10037     last-col,~
10038     last-row,~
10039     left-margin,~
10040     light-syntax,~
10041     light-syntax-expanded,~
10042     name,~
10043     no-cell-nodes,~
10044     nullify-dots,~
10045     pgf-node-code,~
10046     renew-dots,~
10047     respect-arraystretch,~
10048     right-margin,~
10049     rounded-corners,~
10050     rules~(with~the~subkeys~'color'~and~'width'),~
10051     small,~
10052     t,~
10053     vlines,~
10054     xdots/color,~
10055     xdots/shorten-start,~
10056     xdots/shorten-end,~
10057     xdots/shorten-and~
10058     xdots/line-style.
10059 }

```

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`).

```

10060 \@@_msg_new:nmn { Unknown~key~for~NiceMatrix }
10061 {
10062   Unknown~key.\\
10063   The~key~'\l_keys_key_str'~is~unknown~for~the~
10064   \@@_full_name_env:. \\
10065   That~key~will~be~ignored. \\
10066   \c_@@_available_keys_str
10067 }
10068 {
10069   The~available~keys~are~(in~alphabetic~order):~
10070   &-in-blocks,~
10071   ampersand-in-blocks,~
10072   b,~
10073   baseline,~
10074   c,~
10075   cell-space-bottom-limit,~
10076   cell-space-limits,~
10077   cell-space-top-limit,~
10078   code-after,~
10079   code-for-first-col,~
10080   code-for-first-row,~
10081   code-for-last-col,~
10082   code-for-last-row,~
10083   color-inside,~
10084   columns-type,~
10085   columns-width,~
10086   corners,~
10087   create-extra-nodes,~
10088   create-medium-nodes,~
10089   create-large-nodes,~
10090   extra-left-margin,~
10091   extra-right-margin,~
10092   first-col,~
10093   first-row,~
10094   hlines,~
10095   hvlines,~

```



```

10096   hvlines-except-borders,~
10097   l,~
10098   last-col,~
10099   last-row,~
10100   left-margin,~
10101   light-syntax,~
10102   light-syntax-expanded,~
10103   name,~
10104   no-cell-nodes,~
10105   nullify-dots,~
10106   pgf-node-code,~
10107   r,~
10108   renew-dots,~
10109   respect-arraystretch,~
10110   right-margin,~
10111   rounded-corners,~
10112   rules~(with~the~subkeys~'color'~and~'width'),~
10113   small,~
10114   t,~
10115   vlines,~
10116   xdots/color,~
10117   xdots/shorten-start,~
10118   xdots/shorten-end,~
10119   xdots/shorten-and~
10120   xdots/line-style.
10121 }
10122 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10123 {
10124   Unknown~key.\\
10125   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10126   \{NiceTabular\}. \\
10127   That~key~will~be~ignored. \\
10128   \c_@@_available_keys_str
10129 }
10130 {
10131   The~available~keys~are~(in~alphabetic~order):~
10132   &~in~blocks,~
10133   ampersand~in~blocks,~
10134   b,~
10135   baseline,~
10136   c,~
10137   caption,~
10138   cell-space-bottom-limit,~
10139   cell-space-limits,~
10140   cell-space-top-limit,~
10141   code~after,~
10142   code~for~first~col,~
10143   code~for~first~row,~
10144   code~for~last~col,~
10145   code~for~last~row,~
10146   color~inside,~
10147   columns~width,~
10148   corners,~
10149   custom~line,~
10150   create~extra~nodes,~
10151   create~medium~nodes,~
10152   create~large~nodes,~
10153   extra~left~margin,~
10154   extra~right~margin,~
10155   first~col,~
10156   first~row,~
10157   hlines,~
10158   hvlines,~

```

```

10159   hvlines-except-borders,~
10160   label,~
10161   last-col,~
10162   last-row,~
10163   left-margin,~
10164   light-syntax,~
10165   light-syntax-expanded,~
10166   name,~
10167   no-cell-nodes,~
10168   notes~(several~subkeys),~
10169   nullify-dots,~
10170   pgf-node-code,~
10171   renew-dots,~
10172   respect-arraystretch,~
10173   right-margin,~
10174   rounded-corners,~
10175   rules~(with~the~subkeys~'color'~and~'width'),~
10176   short-caption,~
10177   t,~
10178   tabularnote,~
10179   vlines,~
10180   xdots/color,~
10181   xdots/shorten-start,~
10182   xdots/shorten-end,~
10183   xdots/shorten-and~
10184   xdots/line-style.
10185 }

10186 \@@_msg_new:nnn { Duplicate-name }
10187 {
10188   Duplicate~name.\\
10189   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10190   the~same~environment~name~twice.~You~can~go~on,~but,~
10191   maybe,~you~will~have~incorrect~results~especially~
10192   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10193   message~again,~use~the~key~'allow-duplicate-names'~in~
10194   '\token_to_str:N \NiceMatrixOptions'.\\
10195   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10196     { For~a~list~of~the~names~already~used,~type-H~<return>. }
10197 }
10198 {
10199   The~names~already~defined~in~this~document~are:~
10200   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10201 }

10202 \@@_msg_new:nn { Option-auto-for-columns-width }
10203 {
10204   Erroneous~use.\\
10205   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10206   That~key~will~be~ignored.
10207 }

10208 \@@_msg_new:nn { NiceTabularX~without~X }
10209 {
10210   NiceTabularX~without~X.\\
10211   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10212   However,~you~can~go~on.
10213 }

10214 \@@_msg_new:nn { Preamble~forgotten }
10215 {
10216   Preamble~forgotten.\\
10217   You~have~probably~forgotten~the~preamble~of~your~
10218   \@@_full_name_env:. \\
10219   This~error~is~fatal.
10220 }

```

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>	18
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>	73
13	The environment <code>{NiceMatrix}</code> and its variants	90
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
15	After the construction of the array	93
16	We draw the dotted lines	99
17	The actual instructions for drawing the dotted lines with Tikz	113
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	126
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	141
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	158
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	188
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

31	The keyword <code>\CodeAfter</code>	192
32	The delimiters in the preamble	192
33	The command <code>\SubMatrix</code>	194
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	202
35	The command <code>TikzEveryCell</code>	205
36	The command <code>\ShowCellNames</code>	207
37	We process the options at package loading	208
38	About the package underscore	210
39	Error messages of the package	210