The code of the package nicematrix[*]

F. Pantigny
`fpantigny@wanadoo.fr`

March 4, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registred for this package.
See: `http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf`
$<$@@=nicematrix$>$

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

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

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

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

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

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

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

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

```
34 \cs_new_protected:Npn \@@_error_or_warning:n
35   { \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".

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


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

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

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


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


68 \hook_gput_code:nnn { begindocument / end } { . }
69   { \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).

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

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

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

3 Technical definitions

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

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

```
97  \tl_new:N \l_@@_argspec_tl
```

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

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

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.

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

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

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

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

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

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.

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

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

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

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.

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

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

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

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

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

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

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.

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

Idem for `\CT@drs@`.

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

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.

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

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

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

---

[1]See question 99041 on TeX StackExchange.

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

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

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.

```
214  \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`.

```
215    { \@@_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*.

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

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).

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

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

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

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

```
240  \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

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

7

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

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

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

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

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

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

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.

```
278  \int_new:N \g_@@_env_int
```

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

```
279  \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.

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

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

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

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

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

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

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

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

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

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

```
290 \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.

```
291 \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).

```
292 \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.).

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

```
304 \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`.

```
305 \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.

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

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

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

```
309 \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).

```
310 \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).

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

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

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

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag (the `X` columns of nicematrix are inspired by those of tabularx).

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

```
315 \bool_new:N \g_@@_caption_finished_bool
```

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

```
316 \bool_new:N \l_@@_no_cell_nodes_bool
```

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The 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 particuler, 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 contains an ampersand (`&`) 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 weigth 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
```

```
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 create 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).

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

```
390 \seq_new:N \g_@@_future_pos_of_blocks_seq
```

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

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

```
391 \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.

```
392 \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).

```
393 \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).

```
394 \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.

```
395 \bool_new:N \l_@@_width_used_bool
```

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

```
396 \seq_new:N \g_@@_multicolumn_cells_seq
397 \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`).

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

The following counters will be used when drawing the rules.

```
402 \int_new:N \l_@@_start_int
403 \int_set_eq:NN \l_@@_start_int \c_one_int
404 \int_new:N \l_@@_end_int
405 \int_new:N \l_@@_local_start_int
406 \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.

```
407 \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).

```
408 \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`.

```
409 \tl_new:N \l_@@_fill_tl
410 \tl_new:N \l_@@_opacity_tl
411 \tl_new:N \l_@@_draw_tl
412 \seq_new:N \l_@@_tikz_seq
413 \clist_new:N \l_@@_borders_clist
414 \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.

```
415 \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`.

```
416 \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.

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

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

418 `\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`).

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

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

423 `\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).

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

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

425 `\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).

426 `\bool_new:N \l_@@_vlines_block_bool`
427 `\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.

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

429 `\dim_new:N \l_@@_submatrix_extra_height_dim`
430 `\dim_new:N \l_@@_submatrix_left_xshift_dim`
431 `\dim_new:N \l_@@_submatrix_right_xshift_dim`
432 `\clist_new:N \l_@@_hlines_clist`
433 `\clist_new:N \l_@@_vlines_clist`
434 `\clist_new:N \l_@@_submatrix_hlines_clist`
435 `\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}`).

436 `\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.

437 `\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`).

438 `\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.

  ```
  439    \int_new:N \l_@@_first_row_int
  440    \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.

  ```
  441    \int_new:N \l_@@_first_col_int
  442    \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).

  ```
  443    \int_new:N \l_@@_last_row_int
  444    \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]

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

  Idem for `\l_@@_last_col_without_value_bool`

  ```
  446    \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.

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

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

---

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

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

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

449   `\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`).

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

Some utilities

451 `\cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop`
452   `{`

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

453   `\cs_set_nopar:Npn \l_tmpa_tl { #1 }`
454   `\cs_set_nopar:Npn \l_tmpb_tl { #2 }`
455   `}`

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.

456 `\cs_new_protected:Npn \@@_expand_clist:N #1`
457   `{`
458   `\clist_if_in:NnF #1 { all }`
459   `{`
460   `\clist_clear:N \l_tmpa_clist`
461   `\clist_map_inline:Nn #1`
462   `{`

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

463   `\tl_if_in:nnTF { ##1 } { - }`
464   `{ \@@_cut_on_hyphen:w ##1 \q_stop }`
465   `{`

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

466   `\cs_set_nopar:Npn \l_tmpa_tl { ##1 }`
467   `\cs_set_nopar:Npn \l_tmpb_tl { ##1 }`
468   `}`
469   `\int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl`
470   `{ \clist_put_right:Nn \l_tmpa_clist { ####1 } }`
471   `}`
472   `\tl_set_eq:NN #1 \l_tmpa_clist`
473   `}`
474   `}`

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.

475 `\hook_gput_code:nnn { begindocument } { . }`
476   `{`
477   `\dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }`
478   `\dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }`
479   `\dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }`
480   `}`

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 ot 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.

```
481 \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.

```
482 \int_new:N \g_@@_tabularnote_int
483 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

484 \seq_new:N \g_@@_notes_seq
485 \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.

```
486 \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.

```
487 \seq_new:N \l_@@_notes_labels_seq
488 \newcounter { nicematrix_draft }
```

---

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

```
489 \cs_new_protected:Npn \@@_notes_format:n #1
490   {
491     \setcounter { nicematrix_draft } { #1 }
492     \@@_notes_style:n { nicematrix_draft }
493   }
```

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

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

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

```
495 \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.

```
496 \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.

```
497 \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).

```
498 \hook_gput_code:nnn { begindocument } { . }
499   {
500     \IfPackageLoadedTF { enumitem }
501       {
```

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.

```
502         \newlist { tabularnotes } { enumerate } { 1 }
503         \setlist [ tabularnotes ]
504           {
505             topsep = 0pt ,
506             noitemsep ,
507             leftmargin = * ,
508             align = left ,
509             labelsep = 0pt ,
510             label =
511               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
512           }
513         \newlist { tabularnotes* } { enumerate* } { 1 }
514         \setlist [ tabularnotes* ]
515           {
516             afterlabel = \nobreak ,
517             itemjoin = \quad ,
518             label =
519               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } } }
520           }
```

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.

```
521         \NewDocumentCommand \tabularnote { o m }
522           {
523             \bool_lazy_or:nnT { \cs_if_exist_p:N \@captype } \l_@@_in_env_bool
```

```
524            {
525              \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
526                { \@@_error:n { tabularnote~forbidden } }
527                {
528                  \bool_if:NTF \l_@@_in_caption_bool
529                    \@@_tabularnote_caption:nn
530                    \@@_tabularnote:nn
531                  { #1 } { #2 }
532                }
533            }
534          }
535        }
536        {
537          \NewDocumentCommand \tabularnote { o m }
538            {
539              \@@_error_or_warning:n { enumitem~not~loaded }
540              \@@_gredirect_none:n { enumitem~not~loaded }
541            }
542        }
543    }
544  \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
545    { \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`.

```
546  \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
547    {
```

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).

```
548      \int_zero:N \l_tmpa_int
549      \bool_if:NT \l_@@_notes_detect_duplicates_bool
550        {
```

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`.

```
551          \int_zero:N \l_tmpb_int
552          \seq_map_indexed_inline:Nn \g_@@_notes_seq
553            {
554              \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
555              \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
556                {
557                  \tl_if_novalue:nTF { #1 }
558                    { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
559                    { \int_set:Nn \l_tmpa_int { ##1 }  }
560                  \seq_map_break:
561                }
562            }
563          \int_if_zero:nF \l_tmpa_int
564            { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
565        }
566      \int_if_zero:nT \l_tmpa_int
567        {
```

```
568            \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
569            \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
570          }
571        \seq_put_right:Ne \l_@@_notes_labels_seq
572          {
573            \tl_if_novalue:nTF { #1 }
574              {
575                \@@_notes_format:n
576                  {
577                    \int_eval:n
578                      {
579                        \int_if_zero:nTF \l_tmpa_int
580                          \c@tabularnote
581                          \l_tmpa_int
582                      }
583                  }
584              }
585              { #1 }
586          }
587        \peek_meaning:NF \tabularnote
588          {
```

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.

```
589            \hbox_set:Nn \l_tmpa_box
590              {
```

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

```
591                \@@_notes_label_in_tabular:n
592                  {
593                    \seq_use:Nnnn
594                      \l_@@_notes_labels_seq { , } { , } { , }
595                  }
596              }
```

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

```
597            \int_gdecr:N \c@tabularnote
598            \int_set_eq:NN \l_tmpa_int \c@tabularnote
```

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

```
599            \int_gincr:N \g_@@_tabularnote_int
600            \refstepcounter { tabularnote }
601            \int_compare:nNnT \l_tmpa_int = \c@tabularnote
602              { \int_gincr:N \c@tabularnote }
603            \seq_clear:N \l_@@_notes_labels_seq
604            \bool_lazy_or:nnTF
605              { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
606              { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
607              {
608                \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).

```
609                \skip_horizontal:n { \box_wd:N \l_tmpa_box }
610              }
611              { \box_use:N \l_tmpa_box }
612          }
613      }
```

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`.

```
614 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
615   {
616     \bool_if:NTF \g_@@_caption_finished_bool
617       {
618         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
619           { \int_gzero:N \c@tabularnote }
```

Now, we try to detect duplicate notes in the caption. Be careful! We must put `\tl_if_in:NnF` and not `\tl_if_in:NnT`!

```
620         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
621           { \@@_error:n { Identical~notes~in~caption } }
622       }
623       {
```

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

```
624         \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
625           {
```

Now, we know that are in the second composition of the caption since we are reading a tabular note which has yet been read. Now, the value of `\g_@@_notes_caption_int` won't change anymore: it's the number of uses *without optional argument* of the command `\tabularnote` in the caption.

```
626             \bool_gset_true:N \g_@@_caption_finished_bool
627             \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
628             \int_gzero:N \c@tabularnote
629           }
630           { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
631       }
```

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!

```
632     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
633     \seq_put_right:Ne \l_@@_notes_labels_seq
634       {
635         \tl_if_novalue:nTF { #1 }
636           { \@@_notes_format:n { \int_use:N \c@tabularnote } }
637           { #1 }
638       }
639     \peek_meaning:NF \tabularnote
640       {
641         \@@_notes_label_in_tabular:n
642           { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
643         \seq_clear:N \l_@@_notes_labels_seq
644       }
645   }
646 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
647   { \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.

```
648  \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
649    {
650      \begin { pgfscope }
651      \pgfset
652        {
653          inner~sep = \c_zero_dim ,
654          minimum~size = \c_zero_dim
655        }
656      \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
657      \pgfnode
658        { rectangle }
659        { center }
660        {
661          \vbox_to_ht:nn
662            { \dim_abs:n { #5 - #3 } }
663            {
664              \vfill
665              \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
666            }
667        }
668        { #1 }
669        { }
670      \end { pgfscope }
671    }
```

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.

```
672  \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
673    {
674      \begin { pgfscope }
675      \pgfset
676        {
677          inner~sep = \c_zero_dim ,
678          minimum~size = \c_zero_dim
679        }
680      \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
681      \pgfpointdiff { #3 } { #2 }
682      \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
683      \pgfnode
684        { rectangle }
685        { center }
686        {
687          \vbox_to_ht:nn
688            { \dim_abs:n \l_tmpb_dim }
689            { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
690        }
691        { #1 }
692        { }
693      \end { pgfscope }
694    }
```

7   The options

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

```
695  \tl_new:N \l_@@_caption_tl
696  \tl_new:N \l_@@_short_caption_tl
697  \tl_new:N \l_@@_label_tl
```

The following parameter corresponds to the key `caption-above` of `\NiceMatrixOptions`. When this paremeter is `true`, the captions of the environments `{NiceTabular}`, specified with the key `caption` are put above the tabular (and below elsewhere).

```
698 \bool_new:N \l_@@_caption_above_bool
```

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

```
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:nn { #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 `^{...}`.

```
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 catched 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    color-inside .code:n =
804      \@@_warning_gredirect_none:n { key~color-inside } ,
805    colortbl-like .code:n =
806      \@@_warning_gredirect_none:n { key~color-inside } ,
807    ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
808    ampersand-in-blocks .default:n = true ,
809    &-in-blocks .meta:n = ampersand-in-blocks ,
810    no-cell-nodes .code:n =
811      \bool_set_true:N \l_@@_no_cell_nodes_bool
812      \cs_set_protected:Npn \@@_node_for_cell:
813        { \set@color \box_use_drop:N \l_@@_cell_box } ,
814    no-cell-nodes .value_forbidden:n = true ,
815    rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
816    rounded-corners .default:n = 4 pt ,
817    custom-line .code:n = \@@_custom_line:n { #1 } ,
818    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
819    rules .value_required:n = true ,
820    standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
821    standard-cline .default:n = true ,
```

```
822    cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
823    cell-space-top-limit .value_required:n = true ,
824    cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
825    cell-space-bottom-limit .value_required:n = true ,
826    cell-space-limits .meta:n =
827      {
828        cell-space-top-limit = #1 ,
829        cell-space-bottom-limit = #1 ,
830      } ,
831    cell-space-limits .value_required:n = true ,
832    xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
833    light-syntax .code:n =
834      \bool_set_true:N \l_@@_light_syntax_bool
835      \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
836    light-syntax .value_forbidden:n = true ,
837    light-syntax-expanded .code:n =
838      \bool_set_true:N \l_@@_light_syntax_bool
839      \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
840    light-syntax-expanded .value_forbidden:n = true ,
841    end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
842    end-of-row .value_required:n = true ,
843    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
844    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
845    last-row .int_set:N = \l_@@_last_row_int ,
846    last-row .default:n = -1 ,
847    code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
848    code-for-first-col .value_required:n = true ,
849    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
850    code-for-last-col .value_required:n = true ,
851    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
852    code-for-first-row .value_required:n = true ,
853    code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
854    code-for-last-row .value_required:n = true ,
855    hlines .clist_set:N = \l_@@_hlines_clist ,
856    vlines .clist_set:N = \l_@@_vlines_clist ,
857    hlines .default:n = all ,
858    vlines .default:n = all ,
859    vlines-in-sub-matrix .code:n =
860      {
861        \tl_if_single_token:nTF { #1 }
862          {
863            \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
864              { \@@_error:nn { Forbidden~letter } { #1 } }
```

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

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

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

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

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

```
921     c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
922     t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
923     b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
924     baseline .tl_set:N = \l_@@_baseline_tl ,
925     baseline .value_required:n = true ,
926     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)`.

```
927       \str_if_eq:eeTF { #1 } { auto }
928         { \bool_set_true:N \l_@@_auto_columns_width_bool }
929         { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
930     columns-width .value_required:n = true ,
931     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.

```
932        \legacy_if:nF { measuring@ }
933          {
934            \str_set:Ne \l_tmpa_str { #1 }
935            \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
936              { \@@_error:nn { Duplicate~name } { #1 } }
937              { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
938            \str_set_eq:NN \l_@@_name_str \l_tmpa_str
939          } ,
940      name .value_required:n = true ,
941      code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
942      code-after .value_required:n = true ,
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 distincts environments of nicematrix (these names are global and not local to the current TeX scope). However, the option `allow-duplicate-names` disables this feature.

```
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     \m@th % added 2024/11/21
1186     \c_math_toggle_token
```

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

```
1187     \@@_tuning_key_small:
1188   }
1189 \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.

```
1190 \cs_new_protected:Npn \@@_begin_of_row:
1191   {
1192     \int_gincr:N \c@iRow
```

```
1193    \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1194    \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1195    \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1196    \pgfpicture
1197    \pgfrememberpicturepositiononpagetrue
1198    \pgfcoordinate
1199      { \@@_env: - row - \int_use:N \c@iRow - base }
1200      { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1201    \str_if_empty:NF \l_@@_name_str
1202      {
1203        \pgfnodealias
1204          { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1205          { \@@_env: - row - \int_use:N \c@iRow - base }
1206      }
1207    \endpgfpicture
1208  }
```

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.

```
1209 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1210   {
1211     \int_if_zero:nTF \c@iRow
1212       {
1213         \dim_compare:nNnT { \box_dp:N \l_@@_cell_box } > \g_@@_dp_row_zero_dim
1214           { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1215         \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1216           { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1217       }
1218       {
1219         \int_compare:nNnT \c@iRow = \c_one_int
1220           {
1221             \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_one_dim
1222               { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1223           }
1224       }
1225   }
1226 \cs_new_protected:Npn \@@_rotate_cell_box:
1227   {
1228     \box_rotate:Nn \l_@@_cell_box { 90 }
1229     \bool_if:NTF \g_@@_rotate_c_bool
1230       {
1231         \hbox_set:Nn \l_@@_cell_box
1232           {
1233             \m@th % add 2024/11/21
1234             \c_math_toggle_token
1235             \vcenter { \box_use:N \l_@@_cell_box }
1236             \c_math_toggle_token
1237           }
1238       }
1239       {
1240         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1241           {
1242             \vbox_set_top:Nn \l_@@_cell_box
1243               {
1244                 \vbox_to_zero:n { }
1245                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1246                 \box_use:N \l_@@_cell_box
1247               }
1248           }
```

```
1249              }
1250          \bool_gset_false:N \g_@@_rotate_bool
1251          \bool_gset_false:N \g_@@_rotate_c_bool
1252      }
1253  \cs_new_protected:Npn \@@_adjust_size_box:
1254      {
1255          \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1256              {
1257                  \box_set_wd:Nn \l_@@_cell_box
1258                      { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1259                  \dim_gzero:N \g_@@_blocks_wd_dim
1260              }
1261          \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1262              {
1263                  \box_set_dp:Nn \l_@@_cell_box
1264                      { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1265                  \dim_gzero:N \g_@@_blocks_dp_dim
1266              }
1267          \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1268              {
1269                  \box_set_ht:Nn \l_@@_cell_box
1270                      { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1271                  \dim_gzero:N \g_@@_blocks_ht_dim
1272              }
1273      }
1274  \cs_new_protected:Npn \@@_cell_end:
1275      {
```

The following command is nullified in the tabulars.

```
1276          \@@_tuning_not_tabular_end:
1277          \hbox_set_end:
1278          \@@_cell_end_i:
1279      }
1280  \cs_new_protected:Npn \@@_cell_end_i:
1281      {
```

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.

```
1282          \g_@@_cell_after_hook_tl
1283          \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1284          \@@_adjust_size_box:
1285          \box_set_ht:Nn \l_@@_cell_box
1286              { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1287          \box_set_dp:Nn \l_@@_cell_box
1288              { \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").

```
1289          \@@_update_max_cell_width:
```

The following computations are for the "first row" and the "last row".

```
1290          \@@_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:`

38

- 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.

```
1291    \bool_if:NTF \g_@@_empty_cell_bool
1292      { \box_use_drop:N \l_@@_cell_box }
1293      {
1294        \bool_if:NTF \g_@@_not_empty_cell_bool
1295          \@@_print_node_cell:
1296          {
1297            \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1298              \@@_print_node_cell:
1299              { \box_use_drop:N \l_@@_cell_box }
1300          }
1301      }
1302    \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1303      { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1304    \bool_gset_false:N \g_@@_empty_cell_bool
1305    \bool_gset_false:N \g_@@_not_empty_cell_bool
1306  }
```

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

```
1307  \cs_new_protected:Npn \@@_update_max_cell_width:
1308    {
1309      \dim_gset:Nn \g_@@_max_cell_width_dim
1310        { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } } }
1311    }
```

The following variant of `\@@_cell_end:` is only for the columns of type `w{s}{...}` or `W{s}{...}` (which use the horizontal alignement key `s` of `\makebox`).

```
1312  \cs_new_protected:Npn \@@_cell_end_for_w_s:
1313    {
1314      \@@_math_toggle:
1315      \hbox_set_end:
1316      \bool_if:NF \g_@@_rotate_bool
1317        {
1318          \hbox_set:Nn \l_@@_cell_box
1319            {
1320              \makebox [ \l_@@_col_width_dim ] [ s ]
1321                { \hbox_unpack_drop:N \l_@@_cell_box }
1322            }
1323        }
1324      \@@_cell_end_i:
1325    }
```

```
1326  \pgfset
1327    {
1328      nicematrix / cell-node /.style =
1329        {
1330          inner~sep = \c_zero_dim ,
1331          minimum~width = \c_zero_dim
1332        }
1333    }
```

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

```
1334 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1335 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1336   {
1337     \use:c
1338       {
1339         __siunitx_table_align_
1340         \bool_if:NTF \l__siunitx_table_text_bool
1341           \l__siunitx_table_align_text_tl
1342           \l__siunitx_table_align_number_tl
1343         :n
1344       }
1345     { #1 }
1346   }
1347 \cs_new_protected:Npn \@@_print_node_cell:
1348   { \socket_use:nn { nicematrix / siunitx-wrap } { \@@_node_for_cell: } }
```

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

```
1349 \cs_new_protected:Npn \@@_node_for_cell:
1350   {
1351     \pgfpicture
1352     \pgfsetbaseline \c_zero_dim
1353     \pgfrememberpicturepositiononpagetrue
1354     \pgfset { nicematrix / cell-node }
1355     \pgfnode
1356       { rectangle }
1357       { base }
1358       {
```

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

```
1359         \set@color
1360         \box_use_drop:N \l_@@_cell_box
1361       }
1362       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1363       { \l_@@_pgf_node_code_tl }
1364     \str_if_empty:NF \l_@@_name_str
1365       {
1366         \pgfnodealias
1367           { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1368           { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1369       }
1370     \endpgfpicture
1371   }
```

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.

```
1372 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1373   {
1374     \cs_new_protected:Npn \@@_patch_node_for_cell:
1375       {
1376         \hbox_set:Nn \l_@@_cell_box
1377           {
1378             \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1379             \hbox_overlap_left:n
1380               {
1381                 \pgfsys@markposition
1382                   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
```

I don't know why the following adjustement 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.

```
1383                   #1
1384                 }
1385             \box_use:N \l_@@_cell_box
1386             \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1387             \hbox_overlap_left:n
1388               {
1389                 \pgfsys@markposition
1390                   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1391                 #1
1392               }
1393           }
1394         }
1395     }
```

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

```
1396 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1397   {
1398     \@@_patch_node_for_cell:n
1399       { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1400   }
1401   { \@@_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}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 \cdots\cdots\cdots\cdots 6 \\ 7 \cdots\cdots\cdots\cdots \end{pmatrix}$$

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).

```
1402 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1403   {
1404     \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1405       { g_@@_ #2 _ lines _ tl }
1406       {
1407         \use:c { @@ _ draw _ #2 : nnn }
1408           { \int_use:N \c@iRow }
1409           { \int_use:N \c@jCol }
1410           { \exp_not:n { #3 } }
1411       }
1412   }
```

```
1413 \cs_generate_variant:Nn \@@_array:n { o }
1414 \cs_new_protected:Npn \@@_array:n
1415   {
1416 %     \begin{macrocode}
1417     \dim_set:Nn \col@sep
```

```
1418        { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1419      \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1420        { \cs_set_nopar:Npn \@haligonto { } }
1421        { \cs_set_nopar:Npe \@haligonto { to \dim_use:N \l_@@_tabular_width_dim } } }
```

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

```
1422        \@tabarray
```

\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose the \array (of array) with the option t and the right translation will be done further. Remark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here. \str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).

```
1423        [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1424      }
```

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.

```
1425  \bool_if:nTF
1426      { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1427      { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1428      { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

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

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

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

```
1464 \cs_new_protected:Npn \@@_in_everycr:
1465   {
1466     \bool_if:NT \c_@@_recent_array_bool
1467       {
1468         \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1469         \tbl_update_cell_data_for_next_row:
1470       }
1471     \int_gzero:N \c@jCol
1472     \bool_gset_false:N \g_@@_after_col_zero_bool
1473     \bool_if:NF \g_@@_row_of_col_done_bool
1474       {
1475         \@@_create_row_node:
```

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

```
1476        \clist_if_empty:NF \l_@@_hlines_clist
1477          {
1478            \str_if_eq:eeF \l_@@_hlines_clist { all }
1479              {
1480                \clist_if_in:NeT
1481                  \l_@@_hlines_clist
1482                  { \int_eval:n { \c@iRow + 1 } }
1483              }
1484              {
```

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.

```
1485                \int_compare:nNnT \c@iRow > { -1 }
1486                  {
1487                    \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1488                      { \hrule height \arrayrulewidth width \c_zero_dim }
1489                  }
1490              }
1491          }
1492       }
1493  }
```

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

```
1494 \cs_set_protected:Npn \@@_renew_dots:
1495   {
1496     \cs_set_eq:NN \ldots \@@_Ldots
1497     \cs_set_eq:NN \cdots \@@_Cdots
1498     \cs_set_eq:NN \vdots \@@_Vdots
1499     \cs_set_eq:NN \ddots \@@_Ddots
1500     \cs_set_eq:NN \iddots \@@_Iddots
1501     \cs_set_eq:NN \dots \@@_Ldots
1502     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1503   }
```

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

```
1504 \hook_gput_code:nnn { begindocument } { . }
1505   {
1506     \IfPackageLoadedTF { colortbl }
1507       {
1508         \cs_set_protected:Npn \@@_everycr:
1509           { \CT@everycr { \noalign { \@@_in_everycr: } } }
1510       }
1511       {
1512         \cs_new_protected:Npn \@@_everycr:
1513           { \everycr { \noalign { \@@_in_everycr: } } }
1514       }
1515   }
```

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].

```
1516  \hook_gput_code:nnn { begindocument } { . }
1517    {
1518      \IfPackageLoadedTF { booktabs }
1519        {
1520          \cs_new_protected:Npn \@@_patch_booktabs:
1521            { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1522        }
1523        { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1524    }
```

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`.

```
1525  \cs_new_protected:Npn \@@_some_initialization:
1526    {
1527      \@@_everycr:
1528      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1529      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1530      \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1531      \dim_gzero:N \g_@@_dp_ante_last_row_dim
1532      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1533      \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1534    }
```

```
1535  \cs_new_protected:Npn \@@_pre_array_ii:
1536    {
```

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

```
1537      \int_gzero:N \g_@@_total_X_weight_int
```

```
1538      \@@_expand_clist:N \l_@@_hlines_clist
1539      \@@_expand_clist:N \l_@@_vlines_clist
1540      \@@_patch_booktabs:
1541      \box_clear_new:N \l_@@_cell_box
1542      \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}`).

```
1543      \bool_if:NT \l_@@_small_bool
1544        {
1545          \cs_set_nopar:Npn \arraystretch { 0.47 }
1546          \dim_set:Nn \arraycolsep { 1.45 pt }
```

---

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

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

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

```
1547          \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1548        }


1549      \bool_if:NT \g_@@_recreate_cell_nodes_bool
1550        {
1551          \tl_put_right:Nn \@@_begin_of_row:
1552            {
1553              \pgfsys@markposition
1554                { \@@_env: - row - \int_use:N \c@iRow - base }
1555            }
1556        }
```

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

```
1557      \bool_if:nTF
1558        { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1559        {
1560          \cs_set_nopar:Npn \ar@ialign
1561            {
1562              \bool_if:NT \c_@@_testphase_table_bool
1563                \tbl_init_cell_data_for_table:
1564              \@@_some_initialization:
1565              \dim_zero:N \tabskip
```

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

```
1566              \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1567              \halign
1568            }
1569        }
```

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

```
1570        {
1571          \cs_set_nopar:Npn \ialign
1572            {
1573              \@@_some_initialization:
1574              \dim_zero:N \tabskip
1575              \cs_set_eq:NN \ialign \@@_old_ialign:
1576              \halign
1577            }
1578        }
```

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

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

```
1579        \bool_if:NT \c_@@_revtex_bool
1580          {
1581            \IfPackageLoadedT { colortbl }
1582              { \cs_set_protected:Npn \CT@setup { } }
1583          }
```

We keep in memory the old versions or `\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).

```
1584        \cs_set_eq:NN \@@_old_ldots \ldots
1585        \cs_set_eq:NN \@@_old_cdots \cdots
```

```
1586        \cs_set_eq:NN \@@_old_vdots \vdots
1587        \cs_set_eq:NN \@@_old_ddots \ddots
1588        \cs_set_eq:NN \@@_old_iddots \iddots
1589        \bool_if:NTF \l_@@_standard_cline_bool
1590          { \cs_set_eq:NN \cline \@@_standard_cline }
1591          { \cs_set_eq:NN \cline \@@_cline }
1592        \cs_set_eq:NN \Ldots \@@_Ldots
1593        \cs_set_eq:NN \Cdots \@@_Cdots
1594        \cs_set_eq:NN \Vdots \@@_Vdots
1595        \cs_set_eq:NN \Ddots \@@_Ddots
1596        \cs_set_eq:NN \Iddots \@@_Iddots
1597        \cs_set_eq:NN \Hline \@@_Hline:
1598        \cs_set_eq:NN \Hspace \@@_Hspace:
1599        \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1600        \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1601        \cs_set_eq:NN \Block \@@_Block:
1602        \cs_set_eq:NN \rotate \@@_rotate:
1603        \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1604        \cs_set_eq:NN \dotfill \@@_dotfill:
1605        \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1606        \cs_set_eq:NN \diagbox \@@_diagbox:nn
1607        \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1608        \cs_set_eq:NN \TopRule \@@_TopRule
1609        \cs_set_eq:NN \MidRule \@@_MidRule
1610        \cs_set_eq:NN \BottomRule \@@_BottomRule
1611        \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1612        \cs_set_eq:NN \Hbrace \@@_Hbrace
1613        \cs_set_eq:NN \Vbrace \@@_Vbrace
1614        \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1615          { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1616        \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1617        \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1618        \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1619        \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1620        \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1621          { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1622        \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1623          { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1624        \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:.

```
1625        \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1626        \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1627          { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1628        \@@_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!).

```
1629        \tl_if_exist:NT \l_@@_note_in_caption_tl
1630          {
1631            \tl_if_empty:NF \l_@@_note_in_caption_tl
1632              {
1633                \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1634                \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1635              }
1636          }
```

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).

```
1637        \seq_gclear:N \g_@@_multicolumn_cells_seq
1638        \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).

```
1639        \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`).

```
1640        \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.

```
1641        \int_gzero_new:N \g_@@_col_total_int
1642        \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1643        \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.

```
1644        \tl_gclear_new:N \g_@@_Cdots_lines_tl
1645        \tl_gclear_new:N \g_@@_Ldots_lines_tl
1646        \tl_gclear_new:N \g_@@_Vdots_lines_tl
1647        \tl_gclear_new:N \g_@@_Ddots_lines_tl
1648        \tl_gclear_new:N \g_@@_Iddots_lines_tl
1649        \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1650        \tl_gclear:N \g_nicematrix_code_before_tl
1651        \tl_gclear:N \g_@@_pre_code_before_tl
1652    }
```

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

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

```
1653 \cs_new_protected:Npn \@@_pre_array:
1654    {
1655      \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1656      \int_gzero_new:N \c@iRow
1657      \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1658      \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).

```
1659      \int_compare:nNnT \l_@@_last_row_int = { -1 }
1660        {
1661          \bool_set_true:N \l_@@_last_row_without_value_bool
1662          \bool_if:NT \g_@@_aux_found_bool
1663            { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1664        }
1665      \int_compare:nNnT \l_@@_last_col_int = { -1 }
1666        {
1667          \bool_if:NT \g_@@_aux_found_bool
1668            { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1669        }
```

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".

```
1670        \int_compare:nNnT \l_@@_last_row_int > { -2 }
1671          {
1672            \tl_put_right:Nn \@@_update_for_first_and_last_row:
1673              {
1674                \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1675                  { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1676                \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1677                  { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1678              }
1679          }
```

```
1680        \seq_gclear:N \g_@@_cols_vlism_seq
1681        \seq_gclear:N \g_@@_submatrix_seq
```

Now the `\CodeBefore`.

```
1682        \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.

```
1683        \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1684        \seq_gclear:N \g_@@_future_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1685        \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1686        \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 interfers with the construction of the last row-node of the array. We don't want to create such row-node twice (to avaid warnings or, maybe, errors). That's why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```
1687        \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1688        \@@_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.

```
1689        \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]`).

```
1690        \dim_zero_new:N \l_@@_left_delim_dim
1691        \dim_zero_new:N \l_@@_right_delim_dim
1692        \bool_if:NTF \g_@@_delims_bool
1693          {
```

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

```
1694            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1695            \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1696            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1697            \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1698          }
1699          {
1700            \dim_gset:Nn \l_@@_left_delim_dim
1701              { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1702            \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1703          }
```

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).

```
1704    \hbox_set:Nw \l_@@_the_array_box
1705    \skip_horizontal:N \l_@@_left_margin_dim
1706    \skip_horizontal:N \l_@@_extra_left_margin_dim
1707    \bool_if:NT \c_@@_recent_array_bool
1708      { \UseTaggingSocket { tbl / hmode / begin } }
```

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

```
1709    \m@th
1710    \c_math_toggle_token
1711    \bool_if:NTF \l_@@_light_syntax_bool
1712      { \use:c { @@-light-syntax } }
1713      { \use:c { @@-normal-syntax } }
1714  }
```

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

```
1715 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1716  {
1717    \tl_set:Nn \l_tmpa_tl { #1 }
1718    \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1719      { \@@_rescan_for_spanish:N \l_tmpa_tl }
1720    \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1721    \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...

```
1722    \@@_pre_array:
1723  }
```

9 The \CodeBefore

The following command will be executed if the `\CodeBefore` has to be actually executed (that commmand will be used only once and is present alone only for legibility).

```
1724 \cs_new_protected:Npn \@@_pre_code_before:
1725  {
```

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).

```
1726    \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1727    \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1728    \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1729    \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

Now, we will create all the `col` nodes and `row` nodes with the informations written in the `aux` file. You use the technique described in the page 1247 of `pgfmanual.pdf`, version 3.1.10.

```
1730    \pgfsys@markposition { \@@_env: - position }
1731    \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1732    \pgfpicture
1733    \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1734      \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1735        {
1736          \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1737          \pgfcoordinate { \@@_env: - row - ##1 }
1738            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1739        }
```

Now, the recreation of the `col` nodes.

```
1740      \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1741        {
1742          \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1743          \pgfcoordinate { \@@_env: - col - ##1 }
1744            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1745        }
```

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

```
1746      \@@_create_diag_nodes:
```

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

```
1747      \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1748      \endpgfpicture
```

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

```
1749      \@@_create_blocks_nodes:
1750      \IfPackageLoadedT { tikz }
1751        {
1752          \tikzset
1753            {
1754              every~picture / .style =
1755                { overlay , name~prefix = \@@_env: - }
1756            }
1757        }
1758      \cs_set_eq:NN \cellcolor \@@_cellcolor
1759      \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1760      \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1761      \cs_set_eq:NN \rowcolor \@@_rowcolor
1762      \cs_set_eq:NN \rowcolors \@@_rowcolors
1763      \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1764      \cs_set_eq:NN \arraycolor \@@_arraycolor
1765      \cs_set_eq:NN \columncolor \@@_columncolor
1766      \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1767      \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1768      \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1769      \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1770      \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1771      \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1772    }
```

```
1773  \cs_new_protected:Npn \@@_exec_code_before:
1774    {
```

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...

```
1775      \clist_map_inline:Nn \l_@@_corners_cells_clist
1776        { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
```

```
1777      \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.

```
1778        \@@_add_to_colors_seq:nn { { nocolor } } { }
1779        \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1780        \group_begin:
```

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

```
1781        \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 ASCCI 60) and > are activated and Tikz is not able to solve the problem (even with the Tikz library babel).

```
1782        \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1783          { \@@_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.

```
1784        \exp_last_unbraced:No \@@_CodeBefore_keys:
1785          \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.

```
1786        \@@_actually_color:
1787        \l_@@_code_before_tl
1788        \q_stop
1789      \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1790      \group_end:
1791      \bool_if:NT \g_@@_recreate_cell_nodes_bool
1792        { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1793    }
```

```
1794 \keys_define:nn { nicematrix / CodeBefore }
1795   {
1796     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1797     create-cell-nodes .default:n = true ,
1798     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1799     sub-matrix .value_required:n = true ,
1800     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1801     delimiters / color .value_required:n = true ,
1802     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1803   }
1804 \NewDocumentCommand \@@_CodeBefore_keys: { O { } }
1805   {
1806     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1807     \@@_CodeBefore:w
1808   }
```

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.

```
1809 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1810   {
1811     \bool_if:NT \g_@@_aux_found_bool
1812       {
1813         \@@_pre_code_before:
1814         #1
```

```
1815          }
1816    }
```

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.

```
1817  \cs_new_protected:Npn \@@_recreate_cell_nodes:
1818    {
1819      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1820        {
1821          \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1822          \pgfcoordinate { \@@_env: - row - ##1 - base }
1823            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1824          \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1825            {
1826              \cs_if_exist:cT
1827                { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1828                {
1829                  \pgfsys@getposition
1830                    { \@@_env: - ##1 - ####1 - NW }
1831                    \@@_node_position:
1832                  \pgfsys@getposition
1833                    { \@@_env: - ##1 - ####1 - SE }
1834                    \@@_node_position_i:
1835                  \@@_pgf_rect_node:nnn
1836                    { \@@_env: - ##1 - ####1 }
1837                    { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1838                    { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1839                }
1840            }
1841        }
1842      \int_step_inline:nn \c@iRow
1843        {
1844          \pgfnodealias
1845            { \@@_env: - ##1 - last }
1846            { \@@_env: - ##1 - \int_use:N \c@jCol }
1847        }
1848      \int_step_inline:nn \c@jCol
1849        {
1850          \pgfnodealias
1851            { \@@_env: - last - ##1 }
1852            { \@@_env: - \int_use:N \c@iRow - ##1 }
1853        }
1854      \@@_create_extra_nodes:
1855    }
```

```
1856  \cs_new_protected:Npn \@@_create_blocks_nodes:
1857    {
1858      \pgfpicture
1859      \pgf@relevantforpicturesizefalse
1860      \pgfrememberpicturepositiononpagetrue
1861      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1862        { \@@_create_one_block_node:nnnnn ##1 }
1863      \endpgfpicture
1864    }
```

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]

---

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

```
1865  \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1866    {
1867      \tl_if_empty:nF { #5 }
1868        {
1869          \@@_qpoint:n { col - #2 }
1870          \dim_set_eq:NN \l_tmpa_dim \pgf@x
1871          \@@_qpoint:n { #1 }
1872          \dim_set_eq:NN \l_tmpb_dim \pgf@y
1873          \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1874          \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1875          \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1876          \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1877          \@@_pgf_rect_node:nnnnn
1878            { \@@_env: - #5 }
1879            { \dim_use:N \l_tmpa_dim }
1880            { \dim_use:N \l_tmpb_dim }
1881            { \dim_use:N \l_@@_tmpc_dim }
1882            { \dim_use:N \l_@@_tmpd_dim }
1883        }
1884    }


1885  \cs_new_protected:Npn \@@_patch_for_revtex:
1886    {
1887      \cs_set_eq:NN \@addamp \@addamp@LaTeX
1888      \cs_set_eq:NN \@array \@array@array
1889      \cs_set_eq:NN \@tabular \@tabular@array
1890      \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1891      \cs_set_eq:NN \array \array@array
1892      \cs_set_eq:NN \endarray \endarray@array
1893      \cs_set:Npn \endtabular { \endarray $\egroup} % $
1894      \cs_set_eq:NN \@mkpream \@mkpream@array
1895      \cs_set_eq:NN \@classx \@classx@array
1896      \cs_set_eq:NN \insert@column \insert@column@array
1897      \cs_set_eq:NN \@arraycr \@arraycr@array
1898      \cs_set_eq:NN \@xarraycr \@xarraycr@array
1899      \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1900    }
```

10   The environment {NiceArrayWithDelims}

```
1901  \NewDocumentEnvironment { NiceArrayWithDelims }
1902    { m m O { } m ! O { } t \CodeBefore }
1903    {
1904      \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:

1905      \@@_provide_pgfsyspdfmark:
1906      \bool_if:NT \g_@@_footnote_bool \savenotes
```

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

```
1907        \bgroup

1908        \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1909        \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1910        \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1911        \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1912        \int_gzero:N \g_@@_block_box_int
1913        \dim_zero:N \g_@@_width_last_col_dim
```

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

The command `\CT@arc@` contains the instruction of color for the rules of the array[7]. This command is used by `\CT@arc@` but we use it also for compatibility with colortbl. But we want also to be able to use color for the rules of the array when colortbl is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by colortbl. Of course, we restore the value of `\CT@arc@` at the end of our environment.

```
1923    \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@
```

We deactivate Tikz externalization because we will use PGF pictures with the options overlay and remember picture (or equivalent forms). We deactivate with `\tikzexternaldisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```
1924    \cs_if_exist:NT \tikz@library@external@loaded
1925      {
1926        \tikzexternaldisable
1927        \cs_if_exist:NT \ifstandalone
1928          { \tikzset { external / optimize = false } }
1929      }
```

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

```
1930    \int_gincr:N \g_@@_env_int
1931    \bool_if:NF \l_@@_block_auto_columns_width_bool
1932      { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

The sequence `\g_@@_blocks_seq` will contain the carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks.

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

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

```
1935    \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1936    \seq_gclear:N \g_@@_pos_of_xdots_seq
1937    \tl_gclear_new:N \g_@@_code_before_tl
1938    \tl_gclear:N \g_@@_row_style_tl
```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

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

Now, we prepare the token list for the instructions that we will have to write on the aux file at the end of the environment.

```
1945    \tl_gclear:N \g_@@_aux_tl
1946    \tl_if_empty:NF \g_@@_code_before_tl
1947      {
1948        \bool_set_true:N \l_@@_code_before_bool
1949        \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1950      }
1951    \tl_if_empty:NF \g_@@_pre_code_before_tl
```

---

[7] e.g. `\color[rgb]{0.5,0.5,0}`

```
1952        { \bool_set_true:N \l_@@_code_before_bool }
```

The set of keys is not exactly the same for {NiceArray} and for the variants of {NiceArray} ({pNiceArray}, {bNiceArray}, etc.) because, for {NiceArray}, we have the options t, c, b and baseline.

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


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

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

```
1958        \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1959      }
```

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

```
1960      {
1961        \bool_if:NTF \l_@@_light_syntax_bool
1962          { \use:c { end @@-light-syntax } }
1963          { \use:c { end @@-normal-syntax } }
1964        \c_math_toggle_token
1965        \skip_horizontal:N \l_@@_right_margin_dim
1966        \skip_horizontal:N \l_@@_extra_right_margin_dim

1968        % awful workaround
1969        \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1970          {
1971            \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1972              {
1973                \skip_horizontal:N - \l_@@_columns_width_dim
1974                \bool_if:NTF \l_@@_tabular_bool
1975                  { \skip_horizontal:n { - 2 \tabcolsep } }
1976                  { \skip_horizontal:n { - 2 \arraycolsep } }
1977              }
1978          }
1979        \hbox_set_end:
1980        \bool_if:NT \c_@@_recent_array_bool
1981          { \UseTaggingSocket { tbl / hmode / end } }
```

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

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

```
1982        \bool_if:NT \l_@@_width_used_bool
1983          {
1984            \int_if_zero:nT \g_@@_total_X_weight_int
1985              { \@@_error_or_warning:n { width~without~X~columns } }
1986          }
```

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.

```
1987        \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1988          { \@@_compute_width_X: }
```

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

```
1989        \int_compare:nNnT \l_@@_last_row_int > { -2 }
```

```
1990        {
1991          \bool_if:NF \l_@@_last_row_without_value_bool
1992            {
1993              \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1994                {
1995                  \@@_error:n { Wrong~last~row }
1996                  \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1997                }
1998            }
1999        }
```

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]

```
2000        \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2001        \bool_if:NTF \g_@@_last_col_found_bool
2002          { \int_gdecr:N \c@jCol }
2003          {
2004            \int_compare:nNnT \l_@@_last_col_int > { -1 }
2005              { \@@_error:n { last~col~not~used } }
2006          }
```

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

```
2007        \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2008        \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.).

```
2009        \int_if_zero:nT \l_@@_first_col_int
2010          { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2011        \bool_if:nTF { ! \g_@@_delims_bool }
2012          {
2013            \str_if_eq:eeTF \l_@@_baseline_tl { c }
2014              \@@_use_arraybox_with_notes_c:
2015              {
2016                \str_if_eq:eeTF \l_@@_baseline_tl { b }
2017                  \@@_use_arraybox_with_notes_b:
2018                  \@@_use_arraybox_with_notes:
2019              }
2020          }
```

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).

```
2021        {
2022          \int_if_zero:nTF \l_@@_first_row_int
2023            {
2024              \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2025              \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2026            }
2027            { \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]

```
2028          \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2029            {
2030              \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2031              \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2032            }
```

---

[8]We remind that the potential "first column" (exterior) has the number 0.

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

```
2033              { \dim_zero:N \l_tmpb_dim }
2034          \hbox_set:Nn \l_tmpa_box
2035              {
2036                \m@th % added 2024/11/21
2037                \c_math_toggle_token
2038                \@@_color:o \l_@@_delimiters_color_tl
2039                \exp_after:wN \left \g_@@_left_delim_tl
2040                \vcenter
2041                    {
```

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.

```
2042                    \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2043                    \hbox
2044                      {
2045                        \bool_if:NTF \l_@@_tabular_bool
2046                          { \skip_horizontal:N -\tabcolsep }
2047                          { \skip_horizontal:N -\arraycolsep }
2048                        \@@_use_arraybox_with_notes_c:
2049                        \bool_if:NTF \l_@@_tabular_bool
2050                          { \skip_horizontal:N -\tabcolsep }
2051                          { \skip_horizontal:N -\arraycolsep }
2052                      }
```

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

```
2053                    \skip_vertical:N -\l_tmpb_dim
2054                    \skip_vertical:N \arrayrulewidth
2055                  }
2056              \exp_after:wN \right \g_@@_right_delim_tl
2057              \c_math_toggle_token
2058          }
```

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.

```
2059        \bool_if:NTF \l_@@_delimiters_max_width_bool
2060          {
2061            \@@_put_box_in_flow_bis:nn
2062              \g_@@_left_delim_tl
2063              \g_@@_right_delim_tl
2064          }
2065          \@@_put_box_in_flow:
2066      }
```

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.).

```
2067      \bool_if:NT \g_@@_last_col_found_bool
2068        { \skip_horizontal:N \g_@@_width_last_col_dim }
2069      \bool_if:NT \l_@@_preamble_bool
2070        {
2071          \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2072            { \@@_warning_gredirect_none:n { columns~not~used } }
2073        }
2074      \@@_after_array:
```

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

```
2075      \egroup
```

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

```
2076      \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2077      \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 }  }
2078      \iow_now:Ne \@mainaux
2079        {
2080          \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
```

```
2081          { \exp_not:o \g_@@_aux_tl }
2082        }
2083      \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2084      \bool_if:NT \g_@@_footnote_bool \endsavenotes
2085    }
```

This is the end of the environment {NiceArrayWithDelims}.

The following command will be used only once. We have written that command for legibility. If there is at least one X-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `l_@@_X_columns_dim` will be the width of a column of weight 1. For a X-column of weight n, the width will be `\l_@@_X_columns_dim` multiplied by n.

```
2086 \cs_new_protected:Npn \@@_compute_width_X:
2087    {
2088      \tl_gput_right:Ne \g_@@_aux_tl
2089        {
2090          \bool_set_true:N \l_@@_X_columns_aux_bool
2091          \dim_set:Nn \l_@@_X_columns_dim
2092            {
2093              \dim_compare:nNnTF
2094                {
2095                  \dim_abs:n
2096                    { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2097                }
2098                <
2099                { 0.001 pt }
2100                { \dim_use:N \l_@@_X_columns_dim }
2101                {
2102                  \dim_eval:n
2103                    {
2104                      ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2105                      / \int_use:N \g_@@_total_X_weight_int
2106                      + \l_@@_X_columns_dim
2107                    }
2108                }
2109            }
2110        }
2111    }
```

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.

```
2112 \cs_new_protected:Npn \@@_transform_preamble:
2113    {
2114      \@@_transform_preamble_i:
2115      \@@_transform_preamble_ii:
2116    }

2117 \cs_new_protected:Npn \@@_transform_preamble_i:
2118    {
2119      \int_gzero:N \c@jCol
```

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

```
2120      \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.

```
2121        \bool_gset_false:N \g_tmpb_bool
```

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

```
2122        \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2123        \int_zero:N \l_tmpa_int
2124        \tl_gclear:N \g_@@_array_preamble_tl
2125        \str_if_eq:eeTF \l_@@_vlines_clist { all }
2126          {
2127            \tl_gset:Nn \g_@@_array_preamble_tl
2128              { ! { \skip_horizontal:N \arrayrulewidth } }
2129          }
2130          {
2131            \clist_if_in:NnT \l_@@_vlines_clist 1
2132              {
2133                \tl_gset:Nn \g_@@_array_preamble_tl
2134                  { ! { \skip_horizontal:N \arrayrulewidth } }
2135              }
2136          }
```

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

```
2137        \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2138        \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2139        \@@_replace_columncolor:
2140    }


2141  \hook_gput_code:nnn { begindocument } { . }
2142    {
2143      \IfPackageLoadedTF { colortbl }
2144        {
```

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).

```
2145          \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2146          \cs_new_protected:Npn \@@_replace_columncolor:
2147            {
2148              \regex_replace_all:NnN
2149                \c_@@_columncolor_regex
2150                { \c { @@_columncolor_preamble } }
2151                \g_@@_array_preamble_tl
2152            }
2153        }
2154        {
2155          \cs_new_protected:Npn \@@_replace_columncolor:
2156            { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2157        }
2158    }


2159  \cs_new_protected:Npn \@@_transform_preamble_ii:
2160    {
```

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

```
2161        \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2162          {
2163            \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
```

```
2164            { \bool_gset_true:N \g_@@_delims_bool }
2165          }
2166          { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2167          \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).

```
2168        \int_if_zero:nTF \l_@@_first_col_int
2169          { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2170          {
2171            \bool_if:NF \g_@@_delims_bool
2172              {
2173                \bool_if:NF \l_@@_tabular_bool
2174                  {
2175                    \clist_if_empty:NT \l_@@_vlines_clist
2176                      {
2177                        \bool_if:NF \l_@@_exterior_arraycolsep_bool
2178                          { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } } }
2179                  }
2180              }
2181          }
2182        \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2183          { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2184          {
2185            \bool_if:NF \g_@@_delims_bool
2186              {
2187                \bool_if:NF \l_@@_tabular_bool
2188                  {
2189                    \clist_if_empty:NT \l_@@_vlines_clist
2190                      {
2191                        \bool_if:NF \l_@@_exterior_arraycolsep_bool
2192                          { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } } }
2193                  }
2194              }
2195          }
2196      }
2197    }
```

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).

```
2198    \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2199      {
```

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

```
2200        \bool_if:NF \c_@@_testphase_table_bool
2201          {
2202            \tl_gput_right:Nn \g_@@_array_preamble_tl
2203              { > { \@@_error_too_much_cols: } l }
2204          }
2205      }
2206  }
```

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.

```
2207 \cs_new_protected:Npn \@@_rec_preamble:n #1
2208   {
```

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]

```
2209        \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2210          { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2211          {
```

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

```
2212            \cs_if_exist:cTF { NC @ find @ #1 }
2213              {
2214                \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2215                \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2216              }
2217              {
2218                \str_if_eq:nnTF { #1 } { S }
2219                  { \@@_fatal:n { unknown~column~type~S } }
2220                  { \@@_fatal:nn { unknown~column~type } { #1 } }
2221              }
2222          }
2223      }
```

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

```
2224  \cs_new_protected:Npn \@@_c #1
2225    {
2226      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2227      \tl_gclear:N \g_@@_pre_cell_tl
2228      \tl_gput_right:Nn \g_@@_array_preamble_tl
2229        { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
2230      \int_gincr:N \c@jCol
2231      \@@_rec_preamble_after_col:n
2232    }
2233  \cs_new_protected:Npn \@@_l #1
2234    {
2235      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2236      \tl_gclear:N \g_@@_pre_cell_tl
2237      \tl_gput_right:Nn \g_@@_array_preamble_tl
2238        {
2239          > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2240          l
2241          < \@@_cell_end:
2242        }
2243      \int_gincr:N \c@jCol
2244      \@@_rec_preamble_after_col:n
2245    }
2246  \cs_new_protected:Npn \@@_r #1
2247    {
2248      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2249      \tl_gclear:N \g_@@_pre_cell_tl
2250      \tl_gput_right:Nn \g_@@_array_preamble_tl
2251        {
2252          > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2253          r
2254          < \@@_cell_end:
2255        }
2256      \int_gincr:N \c@jCol
2257      \@@_rec_preamble_after_col:n
2258    }
```

---

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

For ! and @

```
2259 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2260   {
2261     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2262     \@@_rec_preamble:n
2263   }
2264 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }
```

For |

```
2265 \cs_new_protected:cpn { @@ _ | } #1
2266   {
```

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

```
2267     \int_incr:N \l_tmpa_int
2268     \@@_make_preamble_i_i:n
2269   }
2270 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2271   {
2272     \str_if_eq:nnTF { #1 } { | }
2273       { \use:c { @@ _ | } | }
2274       { \@@_make_preamble_i_ii:nn { } #1 }
2275   }
2276 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2277   {
2278     \str_if_eq:nnTF { #2 } { [ }
2279       { \@@_make_preamble_i_ii:nw { #1 } [ }
2280       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2281   }
2282 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2283   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2284 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2285   {
2286     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2287     \tl_gput_right:Ne \g_@@_array_preamble_tl
2288       {
```

Here, the command `\dim_use:N` is mandatory.

```
2289         \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2290       }
2291     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2292       {
2293         \@@_vline:n
2294           {
2295             position = \int_eval:n { \c@jCol + 1 } ,
2296             multiplicity = \int_use:N \l_tmpa_int ,
2297             total-width = \dim_use:N \l_@@_rule_width_dim ,
2298             #2
2299           }
```

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.

```
2300       }
2301     \int_zero:N \l_tmpa_int
2302     \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2303     \@@_rec_preamble:n #1
2304   }


2305 \cs_new_protected:cpn { @@ _ > } #1 #2
2306   {
2307     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2308     \@@_rec_preamble:n
2309   }
```

```
2310  \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.

```
2311  \keys_define:nn { nicematrix / p-column }
2312    {
2313      r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2314      r .value_forbidden:n = true ,
2315      c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2316      c .value_forbidden:n = true ,
2317      l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2318      l .value_forbidden:n = true ,
2319      S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2320      S .value_forbidden:n = true ,
2321      p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2322      p .value_forbidden:n = true ,
2323      t .meta:n = p ,
2324      m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2325      m .value_forbidden:n = true ,
2326      b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2327      b .value_forbidden:n = true
2328    }
```

For p but also b and m.

```
2329  \cs_new_protected:Npn \@@_p #1
2330    {
2331      \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2332      \@@_make_preamble_ii_i:n
2333    }
2334  \cs_set_eq:NN \@@_b \@@_p
2335  \cs_set_eq:NN \@@_m \@@_p

2336  \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2337    {
2338      \str_if_eq:nnTF { #1 } { [ }
2339        { \@@_make_preamble_ii_ii:w [ }
2340        { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2341    }

2342  \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2343    { \@@_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.

```
2344  \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2345    {
```

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).

```
2346      \str_set:Nn \l_@@_hpos_col_str { j }
2347      \@@_keys_p_column:n { #1 }
2348      \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2349    }

2350  \cs_new_protected:Npn \@@_keys_p_column:n #1
2351    { \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.

```
2352  \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2353    {
2354      \use:e
```

63

```
2355        {
2356          \@@_make_preamble_ii_v:nnnnnnnn
2357            { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2358            { \dim_eval:n { #1 } }
2359            {
```

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.

```
2360              \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2361                { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2362                {
```

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

```
2363                  \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2364                    { \str_lowercase:o \l_@@_hpos_col_str }
2365                }
2366              \IfPackageLoadedTF { ragged2e }
2367                {
2368                  \str_case:on \l_@@_hpos_col_str
2369                    {
2370                      c { \exp_not:N \Centering }
2371                      l { \exp_not:N \RaggedRight }
2372                      r { \exp_not:N \RaggedLeft }
2373                    }
2374                }
2375                {
2376                  \str_case:on \l_@@_hpos_col_str
2377                    {
2378                      c { \exp_not:N \centering }
2379                      l { \exp_not:N \raggedright }
2380                      r { \exp_not:N \raggedleft }
2381                    }
2382                }
2383              #3
2384            }
2385            { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2386            { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2387            { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2388            { #2 }
2389            {
2390              \str_case:onF \l_@@_hpos_col_str
2391                {
2392                  { j } { c }
2393                  { si } { c }
2394                }
```

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

```
2395                { \str_lowercase:o \l_@@_hpos_col_str }
2396            }
2397        }
```

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

```
2398      \int_gincr:N \c@jCol
2399      \@@_rec_preamble_after_col:n
2400    }
```

#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*.

```
2401 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2402   {
2403     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2404       {
2405         \tl_gput_right:Nn \g_@@_array_preamble_tl
2406           { > \@@_test_if_empty_for_S: }
2407       }
2408       { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2409     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2410     \tl_gclear:N \g_@@_pre_cell_tl
2411     \tl_gput_right:Nn \g_@@_array_preamble_tl
2412       {
2413         > {
```

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.

```
2414             \dim_set:Nn \l_@@_col_width_dim { #2 }
2415             \bool_if:NT \c_@@_testphase_table_bool
2416               { \tag_struct_begin:n { tag = Div } }
2417             \@@_cell_begin:
```

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

```
2418             \use:c { #7 } [ #1 ] { #2 }
```

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

```
2419             \everypar
2420               {
2421                 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2422                 \everypar { }
2423               }
2424             \bool_if:NT \c_@@_testphase_table_bool \tagpdfparaOn
```

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

```
2425             #3
```

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

```
2426             \g_@@_row_style_tl
2427             \arraybackslash
2428             #5
2429           }
2430         #8
2431         < {
2432             #6
```

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

```
2433             \@finalstrut \@arstrutbox
2434             \use:c { end #7 }
```

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

```
2435             #4
2436             \@@_cell_end:
2437             \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2438           }
2439       }
2440   }
```

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

```
2441 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2442   {
```

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 trigerred, we would have other tokens in the TeX flow (and not `&`).

```
2443     \group_align_safe_begin:
2444     \peek_meaning:NTF &
2445       \@@_the_cell_is_empty:
2446       {
2447         \peek_meaning:NTF \\
2448           \@@_the_cell_is_empty:
2449           {
2450             \peek_meaning:NTF \crcr
2451               \@@_the_cell_is_empty:
2452               \group_align_safe_end:
2453           }
2454       }
2455   }

2456 \cs_new_protected:Npn \@@_the_cell_is_empty:
2457   {
2458     \group_align_safe_end:
2459     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2460       {
```

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

```
2461         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2462         \skip_horizontal:N \l_@@_col_width_dim
2463       }
2464   }

2465 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2466   {
2467     \peek_meaning:NT \__siunitx_table_skip:n
2468       { \bool_gset_true:N \g_@@_empty_cell_bool }
2469   }
```

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.

```
2470 \cs_new_protected:Npn \@@_center_cell_box:
2471   {
```

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

```
2472     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2473       {
2474         \int_compare:nNnT
2475         { \box_ht:N \l_@@_cell_box }
2476         >
```

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).

```
2477         { \box_ht:N \strutbox }
2478         {
2479           \hbox_set:Nn \l_@@_cell_box
2480             {
```

```
2481            \box_move_down:nn
2482              {
2483                ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2484                  + \baselineskip ) / 2
2485              }
2486              { \box_use:N \l_@@_cell_box }
2487          }
2488        }
2489      }
2490    }
```

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

```
2491 \cs_new_protected:Npn \@@_V #1 #2
2492   {
2493     \str_if_eq:nnTF { #1 } { [ }
2494       { \@@_make_preamble_V_i:w [ }
2495       { \@@_make_preamble_V_i:w [ ] { #2 } }
2496   }
2497 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2498   { \@@_make_preamble_V_ii:nn { #1 } }
2499 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2500   {
2501     \str_set:Nn \l_@@_vpos_col_str { p }
2502     \str_set:Nn \l_@@_hpos_col_str { j }
2503     \@@_keys_p_column:n { #1 }
2504     \IfPackageLoadedTF { varwidth }
2505       { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2506       {
2507         \@@_error_or_warning:n { varwidth~not~loaded }
2508         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2509       }
2510   }
```

For `w` and `W`

```
2511 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2512 \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.

```
2513 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2514   {
2515     \str_if_eq:nnTF { #3 } { s }
2516       { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2517       { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2518   }
```

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.

```
2519 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2520   {
2521     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2522     \tl_gclear:N \g_@@_pre_cell_tl
2523     \tl_gput_right:Nn \g_@@_array_preamble_tl
2524       {
2525         > {
2526             \dim_set:Nn \l_@@_col_width_dim { #2 }
2527             \@@_cell_begin:
2528             \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2529           }
```

```
2530        c
2531        < {
2532            \@@_cell_end_for_w_s:
2533            #1
2534            \@@_adjust_size_box:
2535            \box_use_drop:N \l_@@_cell_box
2536        }
2537      }
2538    \int_gincr:N \c@jCol
2539    \@@_rec_preamble_after_col:n
2540  }
```

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

```
2541 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2542  {
2543    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2544    \tl_gclear:N \g_@@_pre_cell_tl
2545    \tl_gput_right:Nn \g_@@_array_preamble_tl
2546      {
2547        > {
```

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.

```
2548            \dim_set:Nn \l_@@_col_width_dim { #4 }
2549            \hbox_set:Nw \l_@@_cell_box
2550            \@@_cell_begin:
2551            \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2552          }
2553        c
2554        < {
2555            \@@_cell_end:
2556            \hbox_set_end:
2557            #1
2558            \@@_adjust_size_box:
2559            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2560        }
2561      }
```

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

```
2562    \int_gincr:N \c@jCol
2563    \@@_rec_preamble_after_col:n
2564  }


2565 \cs_new_protected:Npn \@@_special_W:
2566  {
2567    \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2568      { \@@_warning:n { W~warning } }
2569  }
```

For S (of siunitx).

```
2570 \cs_new_protected:Npn \@@_S #1 #2
2571  {
2572    \str_if_eq:nnTF { #2 } { [ }
2573      { \@@_make_preamble_S:w [ }
2574      { \@@_make_preamble_S:w [ ] { #2 } }
2575  }

2576 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2577  { \@@_make_preamble_S_i:n { #1 } }

2578 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2579  {
2580    \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2581    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
```

```
2582        \tl_gclear:N \g_@@_pre_cell_tl
2583        \tl_gput_right:Nn \g_@@_array_preamble_tl
2584          {
2585            > {
```

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

```
2586                \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2587                \keys_set:nn { siunitx } { #1 }
2588                \@@_cell_begin:
2589                \siunitx_cell_begin:w
2590              }
2591          c
2592          <
2593            {
2594              \siunitx_cell_end:
```

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

```
2595              \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2596                {
2597                  \bool_if:NTF \l__siunitx_table_text_bool
2598                    \bool_set_true:N
2599                    \bool_set_false:N
2600                  \l__siunitx_table_text_bool
2601                }
2602              \@@_cell_end:
2603            }
2604          }
```

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

```
2605        \int_gincr:N \c@jCol
2606        \@@_rec_preamble_after_col:n
2607      }
```

For (, [and \{.
```
2608  \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2609    {
2610      \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.
```
2611      \int_if_zero:nTF \c@jCol
2612        {
2613          \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2614            {
```
In that case, in fact, the first letter of the preamble must be considered as the left delimiter of the array.
```
2615              \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2616              \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2617              \@@_rec_preamble:n #2
2618            }
2619            {
2620              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2621              \@@_make_preamble_iv:nn { #1 } { #2 }
2622            }
2623        }
2624        { \@@_make_preamble_iv:nn { #1 } { #2 } }
2625    }
2626  \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2627  \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
```

```
2628  \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2629    {
2630      \tl_gput_right:Ne \g_@@_pre_code_after_tl
2631        { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2632      \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2633        {
2634          \@@_error:nn { delimiter~after~opening } { #2 }
2635          \@@_rec_preamble:n
2636        }
2637        { \@@_rec_preamble:n #2 }
2638    }
```

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

```
2639  \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2640    { \use:c { @@ _ \token_to_str:N ( } }
```

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

```
2641  \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2642    {
2643      \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2644      \tl_if_in:nnTF { ) ] \} } { #2 }
2645        { \@@_make_preamble_v:nnn #1 #2 }
2646        {
2647          \str_if_eq:nnTF { \@@_stop: } { #2 }
2648            {
2649              \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2650                { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2651                {
2652                  \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2653                  \tl_gput_right:Ne \g_@@_pre_code_after_tl
2654                    { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2655                  \@@_rec_preamble:n #2
2656                }
2657            }
2658            {
2659              \tl_if_in:nnT { ( [ \{ \left } { #2 }
2660                { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2661              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2662                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2663              \@@_rec_preamble:n #2
2664            }
2665        }
2666    }
2667  \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2668  \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2669  \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2670    {
2671      \str_if_eq:nnTF { \@@_stop: } { #3 }
2672        {
2673          \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2674            {
2675              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2676              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2677                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2678              \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2679            }
2680            {
2681              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2682              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2683                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
```

```
2684              \@@_error:nn { double~closing~delimiter } { #2 }
2685            }
2686          }
2687          {
2688            \tl_gput_right:Ne \g_@@_pre_code_after_tl
2689              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2690            \@@_error:nn { double~closing~delimiter } { #2 }
2691            \@@_rec_preamble:n #3
2692          }
2693      }


2694  \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2695    { \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 `@{...}`.

```
2696  \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2697    {
2698      \str_if_eq:nnTF { #1 } { < }
2699        \@@_rec_preamble_after_col_i:n
2700        {
2701          \str_if_eq:nnTF { #1 } { @ }
2702            \@@_rec_preamble_after_col_ii:n
2703            {
2704              \str_if_eq:eeTF \l_@@_vlines_clist { all }
2705                {
2706                  \tl_gput_right:Nn \g_@@_array_preamble_tl
2707                    { ! { \skip_horizontal:N \arrayrulewidth } }
2708                }
2709                {
2710                  \clist_if_in:NeT \l_@@_vlines_clist
2711                    { \int_eval:n { \c@jCol + 1 } }
2712                    {
2713                      \tl_gput_right:Nn \g_@@_array_preamble_tl
2714                        { ! { \skip_horizontal:N \arrayrulewidth } }
2715                    }
2716                }
2717              \@@_rec_preamble:n { #1 }
2718            }
2719        }
2720    }
2721  \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2722    {
2723      \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2724      \@@_rec_preamble_after_col:n
2725    }
```

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.

```
2726  \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2727    {
2728      \str_if_eq:eeTF \l_@@_vlines_clist { all }
2729        {
2730          \tl_gput_right:Nn \g_@@_array_preamble_tl
2731            { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2732        }
2733        {
2734          \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2735            {
2736              \tl_gput_right:Nn \g_@@_array_preamble_tl
2737                { @ { #1 \skip_horizontal:N \arrayrulewidth } }
```

```
2738            }
2739          { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2740        }
2741      \@@_rec_preamble:n
2742    }


2743  \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2744    {
2745      \tl_clear:N \l_tmpa_tl
2746      \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2747      \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2748    }
```

The token \NC@find is at the head of the definition of the columns type done by \newcolumntype.
We wan't that token to be no-op here.

```
2749  \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.

```
2750  \cs_new_protected:Npn \@@_X #1 #2
2751    {
2752      \str_if_eq:nnTF { #2 } { [ }
2753        { \@@_make_preamble_X:w [ }
2754        { \@@_make_preamble_X:w [ ] #2 }
2755    }
2756  \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2757    { \@@_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).

```
2758  \keys_define:nn { nicematrix / X-column }
2759    { 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.

```
2760  \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2761    {
```

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).

```
2762      \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).

```
2763      \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.

```
2764      \int_zero_new:N \l_@@_weight_int
2765      \int_set_eq:NN \l_@@_weight_int \c_one_int
2766      \@@_keys_p_column:n { #1 }
```

The unknown keys are put in \l_tmpa_tl

```
2767      \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2768      \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2769        {
2770          \@@_error_or_warning:n { negative~weight }
2771          \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2772        }
2773      \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).

```
2774        \bool_if:NTF \l_@@_X_columns_aux_bool
2775          {
2776            \@@_make_preamble_ii_iv:nnn
2777              { \l_@@_weight_int \l_@@_X_columns_dim }
2778              { minipage }
2779              { \@@_no_update_width: }
2780          }
2781          {
2782            \tl_gput_right:Nn \g_@@_array_preamble_tl
2783              {
2784                > {
2785                    \@@_cell_begin:
2786                    \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`.

```
2787                    \NotEmpty
```

The following code will nullify the box of the cell.

```
2788                    \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2789                      { \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`.

```
2790                    \begin { minipage } { 5 cm } \arraybackslash
2791                  }
2792                c
2793                < {
2794                    \end { minipage }
2795                    \@@_cell_end:
2796                  }
2797              }
2798            \int_gincr:N \c@jCol
2799            \@@_rec_preamble_after_col:n
2800          }
2801      }
```

```
2802  \cs_new_protected:Npn \@@_no_update_width:
2803    {
2804      \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2805        { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2806    }
```

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

```
2807  \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2808    {
2809      \seq_gput_right:Ne \g_@@_cols_vlism_seq
2810        { \int_eval:n { \c@jCol + 1 } }
2811      \tl_gput_right:Ne \g_@@_array_preamble_tl
2812        { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2813      \@@_rec_preamble:n
2814    }
```

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.

```
2815  \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).

```
2816 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2817   { \@@_fatal:n { Preamble~forgotten } }
2818 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2819 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2820 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2821 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2822 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2823 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox } { @@ _ \token_to_str:N \hline }
```

12 The redefinition of \multicolumn

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

```
2824 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2825   {
```

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

```
2826     \multispan { #1 }
2827     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2828     \begingroup
2829     \bool_if:NT \c_@@_testphase_table_bool
2830       { \tbl_update_multicolumn_cell_data:n { #1 } }
2831     \cs_set_nopar:Npn \@addamp
2832       { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2833     \tl_gclear:N \g_@@_preamble_tl
2834     \@@_make_m_preamble:n #2 \q_stop
```

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

```
2835     \exp_args:No \@mkpream \g_@@_preamble_tl
2836     \@addtopreamble \@empty
2837     \endgroup
2838     \bool_if:NT \c_@@_recent_array_bool
2839       { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2840     \int_compare:nNnT { #1 } > \c_one_int
2841       {
2842         \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2843           { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2844         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2845         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2846           {
2847             {
2848               \int_if_zero:nTF \c@jCol
2849                 { \int_eval:n { \c@iRow + 1 } }
2850                 { \int_use:N \c@iRow }
2851             }
2852             { \int_eval:n { \c@jCol + 1 } }
2853             {
2854               \int_if_zero:nTF \c@jCol
2855                 { \int_eval:n { \c@iRow + 1 } }
2856                 { \int_use:N \c@iRow }
```

```
2857                }
2858              { \int_eval:n { \c@jCol + #1 } }
```

The last argument is for the name of the block

```
2859                { }
2860            }
2861        }
```

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

```
2862        \RenewDocumentCommand \cellcolor { O { } m }
2863          {
2864            \tl_gput_right:Ne \g_@@_pre_code_before_tl
2865              {
2866                \@@_rectanglecolor [ ##1 ]
2867                  { \exp_not:n { ##2 } }
2868                  { \int_use:N \c@iRow - \int_use:N \c@jCol }
2869                  { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2870              }
2871            \ignorespaces
2872          }
```

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

```
2873        \cs_set_nopar:Npn \@sharp { #3 }
2874        \@arstrut
2875        \@preamble
2876        \null
```

We add some lines.

```
2877        \int_gadd:Nn \c@jCol { #1 - 1 }
2878        \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2879          { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2880        \ignorespaces
2881      }
```

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.

```
2882  \cs_new_protected:Npn \@@_make_m_preamble:n #1
2883    {
2884      \str_case:nnF { #1 }
2885        {
2886          c { \@@_make_m_preamble_i:n #1 }
2887          l { \@@_make_m_preamble_i:n #1 }
2888          r { \@@_make_m_preamble_i:n #1 }
2889          > { \@@_make_m_preamble_ii:nn #1 }
2890          ! { \@@_make_m_preamble_ii:nn #1 }
2891          @ { \@@_make_m_preamble_ii:nn #1 }
2892          | { \@@_make_m_preamble_iii:n #1 }
2893          p { \@@_make_m_preamble_iv:nnn t #1 }
2894          m { \@@_make_m_preamble_iv:nnn c #1 }
2895          b { \@@_make_m_preamble_iv:nnn b #1 }
2896          w { \@@_make_m_preamble_v:nnnn { } #1 }
2897          W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2898          \q_stop { }
2899        }
2900        {
2901          \cs_if_exist:cTF { NC @ find @ #1 }
2902            {
2903              \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2904              \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2905            }
2906            {
```

```
2907            \str_if_eq:nnTF { #1 } { S }
2908              { \@@_fatal:n { unknown~column~type~S } }
2909              { \@@_fatal:nn { unknown~column~type } { #1 } } }
2910          }
2911        }
2912    }
```

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

```
2913  \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2914    {
2915      \tl_gput_right:Nn \g_@@_preamble_tl
2916        {
2917          > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2918          #1
2919          < \@@_cell_end:
2920        }
```

We test for the presence of a `<`.

```
2921      \@@_make_m_preamble_x:n
2922    }
```

For `>`, `!` and `@`

```
2923  \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2924    {
2925      \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2926      \@@_make_m_preamble:n
2927    }
```

For `|`

```
2928  \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2929    {
2930      \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2931      \@@_make_m_preamble:n
2932    }
```

For `p`, `m` and `b`

```
2933  \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2934    {
2935      \tl_gput_right:Nn \g_@@_preamble_tl
2936        {
2937          > {
2938              \@@_cell_begin:
2939              \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2940              \mode_leave_vertical:
2941              \arraybackslash
2942              \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2943            }
2944          c
2945          < {
2946              \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2947              \end { minipage }
2948              \@@_cell_end:
2949            }
2950        }
```

We test for the presence of a `<`.

```
2951      \@@_make_m_preamble_x:n
2952    }
```

For `w` and `W`

```
2953  \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2954    {
2955      \tl_gput_right:Nn \g_@@_preamble_tl
2956        {
```

```
2957          > {
2958              \dim_set:Nn \l_@@_col_width_dim { #4 }
2959              \hbox_set:Nw \l_@@_cell_box
2960              \@@_cell_begin:
2961              \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2962          }
2963          c
2964          < {
2965              \@@_cell_end:
2966              \hbox_set_end:
2967              \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2968              #1
2969              \@@_adjust_size_box:
2970              \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2971          }
2972      }
```

We test for the presence of a `<`.

```
2973      \@@_make_m_preamble_x:n
2974  }
```

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

```
2975  \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2976  {
2977      \str_if_eq:nnTF { #1 } { < }
2978          \@@_make_m_preamble_ix:n
2979          { \@@_make_m_preamble:n { #1 } }
2980  }
2981  \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2982  {
2983      \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2984      \@@_make_m_preamble_x:n
2985  }
```

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`).

```
2986  \cs_new_protected:Npn \@@_put_box_in_flow:
2987  {
2988      \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2989      \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2990      \str_if_eq:eeTF \l_@@_baseline_tl { c }
2991          { \box_use_drop:N \l_tmpa_box }
2992          \@@_put_box_in_flow_i:
2993  }
```

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

```
2994  \cs_new_protected:Npn \@@_put_box_in_flow_i:
2995  {
2996      \pgfpicture
2997      \@@_qpoint:n { row - 1 }
2998      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2999      \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3000      \dim_gadd:Nn \g_tmpa_dim \pgf@y
3001      \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).

```
3002          \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3003              {
```

```
3004          \int_set:Nn \l_tmpa_int
3005            {
3006              \str_range:Nnn
3007                \l_@@_baseline_tl
3008                6
3009                { \tl_count:o \l_@@_baseline_tl }
3010            }
3011          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3012        }
3013        {
3014          \str_if_eq:eeTF \l_@@_baseline_tl { t }
3015            { \int_set_eq:NN \l_tmpa_int \c_one_int }
3016            {
3017              \str_if_eq:onTF \l_@@_baseline_tl { b }
3018                { \int_set_eq:NN \l_tmpa_int \c@iRow }
3019                { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3020            }
3021          \bool_lazy_or:nnT
3022            { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3023            { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3024            {
3025              \@@_error:n { bad~value~for~baseline }
3026              \int_set_eq:NN \l_tmpa_int \c_one_int
3027            }
3028          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```

We take into account the position of the mathematical axis.

```
3029          \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3030        }
3031      \dim_gsub:Nn \g_tmpa_dim \pgf@y
```

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

```
3032      \endpgfpicture
3033      \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3034      \box_use_drop:N \l_tmpa_box
3035    }
```

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).

```
3036  \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3037    {
```

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.

```
3038      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3039        {
3040          \int_compare:nNnT \c@jCol > \c_one_int
3041            {
3042              \box_set_wd:Nn \l_@@_the_array_box
3043                { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3044            }
3045        }
```

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

```
3046      \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3047      \bool_if:NT \l_@@_caption_above_bool
3048        {
3049          \tl_if_empty:NF \l_@@_caption_tl
3050            {
```

```
3051            \bool_set_false:N \g_@@_caption_finished_bool
3052            \int_gzero:N \c@tabularnote
3053            \@@_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).

```
3054            \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3055              {
3056                \tl_gput_right:Ne \g_@@_aux_tl
3057                  {
3058                    \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3059                      { \int_use:N \g_@@_notes_caption_int }
3060                  }
3061                \int_gzero:N \g_@@_notes_caption_int
3062              }
3063          }
3064      }
```

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

```
3065        \hbox
3066          {
3067            \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.

```
3068            \@@_create_extra_nodes:
3069            \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3070          }
```

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

```
3071      \bool_lazy_any:nT
3072        {
3073          { ! \seq_if_empty_p:N \g_@@_notes_seq }
3074          { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3075          { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3076        }
3077        \@@_insert_tabularnotes:
3078      \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3079      \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3080      \end { minipage }
3081    }
```


```
3082  \cs_new_protected:Npn \@@_insert_caption:
3083    {
3084      \tl_if_empty:NF \l_@@_caption_tl
3085        {
3086          \cs_if_exist:NTF \@captype
3087            { \@@_insert_caption_i: }
3088            { \@@_error:n { caption~outside~float } }
3089        }
3090    }
```


```
3091  \cs_new_protected:Npn \@@_insert_caption_i:
3092    {
3093      \group_begin:
```

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

```
3094        \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.

```
3095        \IfPackageLoadedT { floatrow }
3096          { \cs_set_eq:NN \@makecaption \FR@makecaption }
3097        \tl_if_empty:NTF \l_@@_short_caption_tl
3098          { \caption }
3099          { \caption [ \l_@@_short_caption_tl ] }
3100          { \l_@@_caption_tl }
```

In some circonstancies (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.

```
3101        \bool_if:NF \g_@@_caption_finished_bool
3102          {
3103            \bool_gset_true:N \g_@@_caption_finished_bool
3104            \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3105            \int_gzero:N \c@tabularnote
3106          }
3107        \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3108        \group_end:
3109      }

3110    \cs_new_protected:Npn \@@_tabularnote_error:n #1
3111      {
3112        \@@_error_or_warning:n { tabularnote~below~the~tabular }
3113        \@@_gredirect_none:n { tabularnote~below~the~tabular }
3114      }

3115    \cs_new_protected:Npn \@@_insert_tabularnotes:
3116      {
3117        \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3118        \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3119        \skip_vertical:N 0.65ex
```

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

```
3120        \group_begin:
3121        \l_@@_notes_code_before_tl
3122        \tl_if_empty:NF \g_@@_tabularnote_tl
3123          {
3124            \g_@@_tabularnote_tl \par
3125            \tl_gclear:N \g_@@_tabularnote_tl
3126          }
```

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.

```
3127        \int_compare:nNnT \c@tabularnote > \c_zero_int
3128          {
3129            \bool_if:NTF \l_@@_notes_para_bool
3130              {
3131                \begin { tabularnotes* }
3132                  \seq_map_inline:Nn \g_@@_notes_seq
3133                    { \@@_one_tabularnote:nn ##1 }
3134                  \strut
3135                \end { tabularnotes* }
```

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

```
3136                \par
```

```
3137              }
3138            {
3139               \tabularnotes
3140                 \seq_map_inline:Nn \g_@@_notes_seq
3141                   { \@@_one_tabularnote:nn ##1 }
3142               \strut
3143               \endtabularnotes
3144            }
3145        }
3146      \unskip
3147      \group_end:
3148      \bool_if:NT \l_@@_notes_bottomrule_bool
3149        {
3150          \IfPackageLoadedTF { booktabs }
3151            {
```

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

```
3152              \skip_vertical:N \aboverulesep
```

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

```
3153              { \CT@arc@ \hrule height \heavyrulewidth }
3154            }
3155            { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3156        }
3157      \l_@@_notes_code_after_tl
3158      \seq_gclear:N \g_@@_notes_seq
3159      \seq_gclear:N \g_@@_notes_in_caption_seq
3160      \int_gzero:N \c@tabularnote
3161    }
```

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.

```
3162  \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3163    {
3164      \tl_if_novalue:nTF { #1 }
3165        { \item }
3166        { \item [ \@@_notes_label_in_list:n { #1 } ] }
3167    }
```

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.

```
3168  \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3169    {
3170      \pgfpicture
3171        \@@_qpoint:n { row - 1 }
3172        \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3173        \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3174        \dim_gsub:Nn \g_tmpa_dim \pgf@y
3175      \endpgfpicture
3176      \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3177      \int_if_zero:nT \l_@@_first_row_int
3178        {
3179          \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3180          \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3181        }
3182      \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3183    }
```

Now, the general case.

```
3184  \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3185    {
```

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

```
3186        \str_if_eq:eeT \l_@@_baseline_tl { t }
3187          { \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`.

```
3188        \pgfpicture
3189        \@@_qpoint:n { row - 1 }
3190        \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3191        \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3192          {
3193            \int_set:Nn \l_tmpa_int
3194              {
3195                \str_range:Nnn
3196                  \l_@@_baseline_tl
3197                  6
3198                  { \tl_count:o \l_@@_baseline_tl }
3199              }
3200            \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3201          }
3202          {
3203            \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3204            \bool_lazy_or:nnT
3205              { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3206              { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3207              {
3208                \@@_error:n { bad~value~for~baseline }
3209                \int_set:Nn \l_tmpa_int 1
3210              }
3211            \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3212          }
3213        \dim_gsub:Nn \g_tmpa_dim \pgf@y
3214        \endpgfpicture
3215        \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3216        \int_if_zero:nT \l_@@_first_row_int
3217          {
3218            \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3219            \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3220          }
3221        \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3222      }
```

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.

```
3223  \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3224    {
```

We will compute the real width of both delimiters used.

```
3225        \dim_zero_new:N \l_@@_real_left_delim_dim
3226        \dim_zero_new:N \l_@@_real_right_delim_dim
3227        \hbox_set:Nn \l_tmpb_box
3228          {
3229            \m@th % added 2024/11/21
3230            \c_math_toggle_token
3231            \left #1
3232            \vcenter
3233              {
3234                \vbox_to_ht:nn
3235                  { \box_ht_plus_dp:N \l_tmpa_box }
3236                  { }
3237              }
3238            \right .
```

```
3239        \c_math_toggle_token
3240      }
3241    \dim_set:Nn \l_@@_real_left_delim_dim
3242      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3243    \hbox_set:Nn \l_tmpb_box
3244      {
3245        \m@th % added 2024/11/21
3246        \c_math_toggle_token
3247        \left .
3248        \vbox_to_ht:nn
3249          { \box_ht_plus_dp:N \l_tmpa_box }
3250          { }
3251        \right #2
3252        \c_math_toggle_token
3253      }
3254    \dim_set:Nn \l_@@_real_right_delim_dim
3255      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3256        \skip_horizontal:N  \l_@@_left_delim_dim
3257        \skip_horizontal:N -\l_@@_real_left_delim_dim
3258        \@@_put_box_in_flow:
3259        \skip_horizontal:N \l_@@_right_delim_dim
3260        \skip_horizontal:N -\l_@@_real_right_delim_dim
3261      }
```

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).

```
3262 \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).

```
3263    {
3264      \peek_remove_spaces:n
3265        {
3266          \peek_meaning:NTF \end
3267            \@@_analyze_end:Nn
3268            {
3269              \@@_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).

```
3270              \@@_array:o \g_@@_array_preamble_tl
3271            }
3272        }
3273    }
3274    {
3275      \@@_create_col_nodes:
3276      \endarray
3277    }
```

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

```
3278 \NewDocumentEnvironment { @@-light-syntax } { b }
3279    {
```

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.

```
3280        \tl_if_empty:nT { #1 }
3281          { \@@_fatal:n { empty~environment } }
3282        \tl_if_in:nnT { #1 } { & }
3283          { \@@_fatal:n { ampersand~in~light-syntax } }
3284        \tl_if_in:nnT { #1 } { \\ }
3285          { \@@_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 catched 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.

```
3286        \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3287      }
```

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.

```
3288      {
3289        \@@_create_col_nodes:
3290        \endarray
3291      }
```

```
3292  \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3293    {
3294        \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).

```
3295        \seq_clear_new:N \l_@@_rows_seq
```

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

```
3296        \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3297        \bool_if:NTF \l_@@_light_syntax_expanded_bool
3298          \seq_set_split:Nee
3299          \seq_set_split:Non
3300          \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3301        \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3302        \tl_if_empty:NF \l_tmpa_tl
3303          { \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.

```
3304        \int_compare:nNnT \l_@@_last_row_int = { -1 }
3305          { \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).

```
3306        \tl_build_begin:N \l_@@_new_body_tl
3307        \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3308        \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3309        \@@_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).

```
3310      \seq_map_inline:Nn \l_@@_rows_seq
3311        {
3312          \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3313          \@@_line_with_light_syntax:n { ##1 }
3314        }
3315      \tl_build_end:N \l_@@_new_body_tl

3316      \int_compare:nNnT \l_@@_last_col_int = { -1 }
3317        {
3318          \int_set:Nn \l_@@_last_col_int
3319            { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3320        }
```

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.

```
3321      \@@_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).

```
3322      \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3323    }
3324  \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3325  \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3326    {
3327      \seq_clear_new:N \l_@@_cells_seq
3328      \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3329      \int_set:Nn \l_@@_nb_cols_int
3330        {
3331          \int_max:nn
3332            \l_@@_nb_cols_int
3333            { \seq_count:N \l_@@_cells_seq }
3334        }
3335      \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3336      \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3337      \seq_map_inline:Nn \l_@@_cells_seq
3338        { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3339    }
```

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`.

```
3340  \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3341    {
3342      \str_if_eq:eeT \g_@@_name_env_str { #2 }
3343        { \@@_fatal:n { empty~environment } }
```

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

```
3344      \end { #2 }
3345    }
```

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`).

```
3346  \cs_new:Npn \@@_create_col_nodes:
3347    {
3348      \crcr
3349      \int_if_zero:nT \l_@@_first_col_int
3350        {
3351          \omit
```

```
3352        \hbox_overlap_left:n
3353          {
3354            \bool_if:NT \l_@@_code_before_bool
3355              { \pgfsys@markposition { \@@_env: - col - 0 } }
3356            \pgfpicture
3357            \pgfrememberpicturepositiononpagetrue
3358            \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3359            \str_if_empty:NF \l_@@_name_str
3360              { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3361            \endpgfpicture
3362            \skip_horizontal:N 2\col@sep
3363            \skip_horizontal:N \g_@@_width_first_col_dim
3364          }
3365      &
3366    }
3367  \omit
```

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

```
3368      \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`).

```
3369      \int_if_zero:nTF \l_@@_first_col_int
3370        {
3371          \bool_if:NT \l_@@_code_before_bool
3372            {
3373              \hbox
3374                {
3375                  \skip_horizontal:N -0.5\arrayrulewidth
3376                  \pgfsys@markposition { \@@_env: - col - 1 }
3377                  \skip_horizontal:N 0.5\arrayrulewidth
3378                }
3379            }
3380          \pgfpicture
3381          \pgfrememberpicturepositiononpagetrue
3382          \pgfcoordinate { \@@_env: - col - 1 }
3383            { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3384          \str_if_empty:NF \l_@@_name_str
3385            { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3386          \endpgfpicture
3387        }
3388        {
3389          \bool_if:NT \l_@@_code_before_bool
3390            {
3391              \hbox
3392                {
3393                  \skip_horizontal:N 0.5\arrayrulewidth
3394                  \pgfsys@markposition { \@@_env: - col - 1 }
3395                  \skip_horizontal:N -0.5\arrayrulewidth
3396                }
3397            }
3398          \pgfpicture
3399          \pgfrememberpicturepositiononpagetrue
3400          \pgfcoordinate { \@@_env: - col - 1 }
3401            { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3402          \str_if_empty:NF \l_@@_name_str
3403            { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3404          \endpgfpicture
3405        }
```

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.

```
3406        \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3407        \bool_if:NF \l_@@_auto_columns_width_bool
3408          { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3409          {
3410            \bool_lazy_and:nnTF
3411              \l_@@_auto_columns_width_bool
3412              { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3413              { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3414              { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3415            \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3416          }
3417        \skip_horizontal:N \g_tmpa_skip
3418        \hbox
3419          {
3420            \bool_if:NT \l_@@_code_before_bool
3421              {
3422                \hbox
3423                  {
3424                    \skip_horizontal:N -0.5\arrayrulewidth
3425                    \pgfsys@markposition { \@@_env: - col - 2 }
3426                    \skip_horizontal:N 0.5\arrayrulewidth
3427                  }
3428              }
3429            \pgfpicture
3430            \pgfrememberpicturepositiononpagetrue
3431            \pgfcoordinate { \@@_env: - col - 2 }
3432              { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3433            \str_if_empty:NF \l_@@_name_str
3434              { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3435            \endpgfpicture
3436          }
```

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.

```
3437        \int_gset_eq:NN \g_tmpa_int \c_one_int
3438        \bool_if:NTF \g_@@_last_col_found_bool
3439          { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3440          { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3441          {
3442            &
3443            \omit
3444            \int_gincr:N \g_tmpa_int
```

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

```
3445            \skip_horizontal:N \g_tmpa_skip
3446            \bool_if:NT \l_@@_code_before_bool
3447              {
3448                \hbox
3449                  {
3450                    \skip_horizontal:N -0.5\arrayrulewidth
3451                    \pgfsys@markposition
3452                      { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3453                    \skip_horizontal:N 0.5\arrayrulewidth
3454                  }
3455              }
```

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

```
3456            \pgfpicture
3457            \pgfrememberpicturepositiononpagetrue
3458            \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3459              { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3460            \str_if_empty:NF \l_@@_name_str
```

```
3461                {
3462                  \pgfnodealias
3463                    { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3464                    { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3465                }
3466            \endpgfpicture
3467          }


3468          &
3469          \omit
```

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

```
3470          \int_if_zero:nT \g_@@_col_total_int
3471            { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3472          \skip_horizontal:N \g_tmpa_skip
3473          \int_gincr:N \g_tmpa_int
3474          \bool_lazy_any:nF
3475            {
3476              \g_@@_delims_bool
3477              \l_@@_tabular_bool
3478              { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3479              \l_@@_exterior_arraycolsep_bool
3480              \l_@@_bar_at_end_of_pream_bool
3481            }
3482            { \skip_horizontal:N -\col@sep }
3483          \bool_if:NT \l_@@_code_before_bool
3484            {
3485              \hbox
3486                {
3487                  \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.

```
3488                  \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3489                    { \skip_horizontal:N -\arraycolsep }
3490                  \pgfsys@markposition
3491                    { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3492                  \skip_horizontal:N 0.5\arrayrulewidth
3493                  \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3494                    { \skip_horizontal:N \arraycolsep }
3495                }
3496            }
3497          \pgfpicture
3498            \pgfrememberpicturepositiononpagetrue
3499            \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3500              {
3501                \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3502                  {
3503                    \pgfpoint
3504                      { - 0.5 \arrayrulewidth - \arraycolsep }
3505                      \c_zero_dim
3506                  }
3507                  { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3508              }
3509            \str_if_empty:NF \l_@@_name_str
3510              {
3511                \pgfnodealias
3512                  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3513                  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3514              }
3515          \endpgfpicture
```

```
3516        \bool_if:NT \g_@@_last_col_found_bool
3517          {
3518            \hbox_overlap_right:n
3519              {
3520                \skip_horizontal:N \g_@@_width_last_col_dim
3521                \skip_horizontal:N \col@sep
3522                \bool_if:NT \l_@@_code_before_bool
3523                  {
3524                    \pgfsys@markposition
3525                      { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3526                  }
3527                \pgfpicture
3528                \pgfrememberpicturepositiononpagetrue
3529                \pgfcoordinate
3530                  { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3531                \pgfpointorigin
3532                \str_if_empty:NF \l_@@_name_str
3533                  {
3534                    \pgfnodealias
3535                      {
3536                        \l_@@_name_str - col
3537                        - \int_eval:n { \g_@@_col_total_int + 1 }
3538                      }
3539                      { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3540                  }
3541                \endpgfpicture
3542              }
3543          }
3544    % \cr
3545      }
```

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

```
3546 \tl_const:Nn \c_@@_preamble_first_col_tl
3547   {
3548     >
3549       {
```

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

```
3550         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3551         \bool_gset_true:N \g_@@_after_col_zero_bool
3552         \@@_begin_of_row:
3553         \hbox_set:Nw \l_@@_cell_box
3554         \@@_math_toggle:
3555         \@@_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".

```
3556         \int_compare:nNnT \c@iRow > \c_zero_int
3557           {
3558             \bool_lazy_or:nnT
3559               { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3560               { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3561               {
3562                 \l_@@_code_for_first_col_tl
3563                 \xglobal \colorlet { nicematrix-first-col } { . }
3564               }
3565           }
3566       }
```

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`.

```
3567       l
```

```
3568        <
3569          {
3570            \@@_math_toggle:
3571            \hbox_set_end:
3572            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3573            \@@_adjust_size_box:
3574            \@@_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.
```
3575            \dim_gset:Nn \g_@@_width_first_col_dim
3576              { \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.
```
3577            \hbox_overlap_left:n
3578              {
3579                \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3580                  \@@_node_for_cell:
3581                  { \box_use_drop:N \l_@@_cell_box }
3582                \skip_horizontal:N \l_@@_left_delim_dim
3583                \skip_horizontal:N \l_@@_left_margin_dim
3584                \skip_horizontal:N \l_@@_extra_left_margin_dim
3585              }
3586            \bool_gset_false:N \g_@@_empty_cell_bool
3587            \skip_horizontal:N -2\col@sep
3588          }
3589      }
```
Here is the preamble for the "last column" (if the user uses the key last-col).
```
3590 \tl_const:Nn \c_@@_preamble_last_col_tl
3591   {
3592      >
3593        {
3594          \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).
```
3595          \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.
```
3596          \bool_gset_true:N \g_@@_last_col_found_bool
3597          \int_gincr:N \c@jCol
3598          \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3599          \hbox_set:Nw \l_@@_cell_box
3600            \@@_math_toggle:
3601            \@@_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".
```
3602          \int_compare:nNnT \c@iRow > \c_zero_int
3603            {
3604              \bool_lazy_or:nnT
3605                { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3606                { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3607                {
3608                  \l_@@_code_for_last_col_tl
3609                  \xglobal \colorlet { nicematrix-last-col } { . }
3610                }
3611            }
3612      }
3613      l
3614        <
3615          {
3616            \@@_math_toggle:
3617            \hbox_set_end:
3618            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
```

```
3619          \@@_adjust_size_box:
3620          \@@_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.

```
3621          \dim_gset:Nn \g_@@_width_last_col_dim
3622            { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3623          \skip_horizontal:N -2\col@sep
```

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

```
3624          \hbox_overlap_right:n
3625            {
3626              \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3627                {
3628                  \skip_horizontal:N \l_@@_right_delim_dim
3629                  \skip_horizontal:N \l_@@_right_margin_dim
3630                  \skip_horizontal:N \l_@@_extra_right_margin_dim
3631                  \@@_node_for_cell:
3632                }
3633            }
3634          \bool_gset_false:N \g_@@_empty_cell_bool
3635        }
3636    }
```

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

```
3637  \NewDocumentEnvironment { NiceArray } { }
3638    {
3639      \bool_gset_false:N \g_@@_delims_bool
3640      \str_if_empty:NT \g_@@_name_env_str
3641        { \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).

```
3642      \NiceArrayWithDelims . .
3643    }
3644    { \endNiceArrayWithDelims }
```

We create the variants of the environment {NiceArrayWithDelims}.

```
3645  \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3646    {
3647      \NewDocumentEnvironment { #1 NiceArray } { }
3648        {
3649          \bool_gset_true:N \g_@@_delims_bool
3650          \str_if_empty:NT \g_@@_name_env_str
3651            { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3652          \@@_test_if_math_mode:
3653          \NiceArrayWithDelims #2 #3
3654        }
3655        { \endNiceArrayWithDelims }
3656    }
3657  \@@_def_env:nnn p ( )
3658  \@@_def_env:nnn b [ ]
3659  \@@_def_env:nnn B \{ \}
3660  \@@_def_env:nnn v | |
3661  \@@_def_env:nnn V \| \|
```

13 The environment {NiceMatrix} and its variants

```
3662 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3663 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3664   {
3665     \bool_set_false:N \l_@@_preamble_bool
3666     \tl_clear:N \l_tmpa_tl
3667     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3668       { \tl_set:Nn \l_tmpa_tl { @ { } } }
3669     \tl_put_right:Nn \l_tmpa_tl
3670       {
3671         *
3672           {
3673             \int_case:nnF \l_@@_last_col_int
3674               {
3675                 { -2 } { \c@MaxMatrixCols }
3676                 { -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).

```
3677              }
3678              { \int_eval:n { \l_@@_last_col_int - 1 } }
3679          }
3680          { #2 }
3681       }
3682     \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3683     \exp_args:No \l_tmpb_tl \l_tmpa_tl
3684   }
3685 \clist_map_inline:nn { p , b , B , v , V }
3686   {
3687     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3688       {
3689         \bool_gset_true:N \g_@@_delims_bool
3690         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3691         \int_if_zero:nT \l_@@_last_col_int
3692           {
3693             \bool_set_true:N \l_@@_last_col_without_value_bool
3694             \int_set:Nn \l_@@_last_col_int { -1 }
3695           }
3696         \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3697         \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3698       }
3699       { \use:c { end #1 NiceArray } }
3700   }
```

We define also an environment {NiceMatrix}

```
3701 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3702   {
3703     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3704     \int_if_zero:nT \l_@@_last_col_int
3705       {
3706         \bool_set_true:N \l_@@_last_col_without_value_bool
3707         \int_set:Nn \l_@@_last_col_int { -1 }
3708       }
3709     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3710     \bool_lazy_or:nnT
3711       { \clist_if_empty_p:N \l_@@_vlines_clist }
3712       { \l_@@_except_borders_bool }
3713       { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3714     \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3715   }
3716   { \endNiceArray }
```

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

```
3717 \cs_new_protected:Npn \@@_NotEmpty:
3718   { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3719 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3720   {
```

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.

```
3721     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3722       { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3723     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3724     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3725     \tl_if_empty:NF \l_@@_short_caption_tl
3726       {
3727         \tl_if_empty:NT \l_@@_caption_tl
3728           {
3729             \@@_error_or_warning:n { short~caption~without~caption }
3730             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3731           }
3732       }
3733     \tl_if_empty:NF \l_@@_label_tl
3734       {
3735         \tl_if_empty:NT \l_@@_caption_tl
3736           { \@@_error_or_warning:n { label~without~caption } }
3737       }
3738     \NewDocumentEnvironment { TabularNote } { b }
3739       {
3740         \bool_if:NTF \l_@@_in_code_after_bool
3741           { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3742           {
3743             \tl_if_empty:NF \g_@@_tabularnote_tl
3744               { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3745             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3746           }
3747       }
3748       { }
3749     \@@_settings_for_tabular:
3750     \NiceArray { #2 }
3751   }
3752   { \endNiceArray }
3753 \cs_new_protected:Npn \@@_settings_for_tabular:
3754   {
3755     \bool_set_true:N \l_@@_tabular_bool
3756     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3757     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3758     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3759   }


3760 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3761   {
3762     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3763     \dim_zero_new:N \l_@@_width_dim
3764     \dim_set:Nn \l_@@_width_dim { #1 }
3765     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3766     \@@_settings_for_tabular:
3767     \NiceArray { #3 }
3768   }
3769   {
3770     \endNiceArray
```

```
3771        \int_if_zero:nT \g_@@_total_X_weight_int
3772          { \@@_error:n { NiceTabularX~without~X } }
3773    }


3774  \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3775    {
3776      \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3777      \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3778      \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3779      \@@_settings_for_tabular:
3780      \NiceArray { #3 }
3781    }
3782    { \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`).

```
3783  \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3784    {
3785      \bool_lazy_all:nT
3786        {
3787          { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3788          \l_@@_hvlines_bool
3789          { ! \g_@@_delims_bool }
3790          { ! \l_@@_except_borders_bool }
3791        }
3792        {
3793          \bool_set_true:N \l_@@_except_borders_bool
3794          \clist_if_empty:NF \l_@@_corners_clist
3795            { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3796          \tl_gput_right:Nn \g_@@_pre_code_after_tl
3797            {
3798              \@@_stroke_block:nnn
3799                {
3800                  rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3801                  draw = \l_@@_rules_color_tl
3802                }
3803                { 1-1 }
3804                { \int_use:N \c@iRow - \int_use:N \c@jCol }
3805            }
3806        }
3807    }


3808  \cs_new_protected:Npn \@@_after_array:
3809    {
```

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.

```
3810        \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3811        \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.

```
3812        \bool_if:NT \g_@@_last_col_found_bool
3813          { \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.

```
3814        \bool_if:NT \l_@@_last_col_without_value_bool
3815          { \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.

```
3816        \bool_if:NT \l_@@_last_row_without_value_bool
3817          { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }


3818        \tl_gput_right:Ne \g_@@_aux_tl
3819          {
3820            \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3821              {
3822                \int_use:N \l_@@_first_row_int ,
3823                \int_use:N \c@iRow ,
3824                \int_use:N \g_@@_row_total_int ,
3825                \int_use:N \l_@@_first_col_int ,
3826                \int_use:N \c@jCol ,
3827                \int_use:N \g_@@_col_total_int
3828              }
3829          }
```

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).

```
3830        \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3831          {
3832            \tl_gput_right:Ne \g_@@_aux_tl
3833              {
3834                \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3835                  { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3836              }
3837          }
3838        \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3839          {
3840            \tl_gput_right:Ne \g_@@_aux_tl
3841              {
3842                \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3843                  { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3844                \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3845                  { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3846              }
3847          }
```

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

```
3848        \@@_create_diag_nodes:
```

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

```
3849        \pgfpicture
3850        \int_step_inline:nn \c@iRow
3851          {
3852            \pgfnodealias
3853              { \@@_env: - ##1 - last }
3854              { \@@_env: - ##1 - \int_use:N \c@jCol }
3855          }
3856        \int_step_inline:nn \c@jCol
3857          {
3858            \pgfnodealias
```

```
3859            { \@@_env: - last - ##1 }
3860            { \@@_env: - \int_use:N \c@iRow - ##1 }
3861        }
3862    \str_if_empty:NF \l_@@_name_str
3863      {
3864        \int_step_inline:nn \c@iRow
3865          {
3866            \pgfnodealias
3867              { \l_@@_name_str - ##1 - last }
3868              { \@@_env: - ##1 - \int_use:N \c@jCol }
3869          }
3870        \int_step_inline:nn \c@jCol
3871          {
3872            \pgfnodealias
3873              { \l_@@_name_str - last - ##1 }
3874              { \@@_env: - \int_use:N \c@iRow - ##1 }
3875          }
3876      }
3877    \endpgfpicture
```

By default, the diagonal lines will be parallelized[11]. 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.

```
3878        \bool_if:NT \l_@@_parallelize_diags_bool
3879          {
3880            \int_gzero_new:N \g_@@_ddots_int
3881            \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.

```
3882            \dim_gzero_new:N \g_@@_delta_x_one_dim
3883            \dim_gzero_new:N \g_@@_delta_y_one_dim
3884            \dim_gzero_new:N \g_@@_delta_x_two_dim
3885            \dim_gzero_new:N \g_@@_delta_y_two_dim
3886          }
3887        \int_zero_new:N \l_@@_initial_i_int
3888        \int_zero_new:N \l_@@_initial_j_int
3889        \int_zero_new:N \l_@@_final_i_int
3890        \int_zero_new:N \l_@@_final_j_int
3891        \bool_set_false:N \l_@@_initial_open_bool
3892        \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.

```
3893        \bool_if:NT \l_@@_small_bool
3894          {
3895            \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3896            \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }
```

The dimensions `\l_@@_xdots_shorten_start_dim` and `\l_@@_xdots_shorten_start_dim` corre-
spond to the options `xdots/shorten-start` and `xdots/shorten-end` available to the user.

```
3897            \dim_set:Nn \l_@@_xdots_shorten_start_dim
3898              { 0.6 \l_@@_xdots_shorten_start_dim }
3899            \dim_set:Nn \l_@@_xdots_shorten_end_dim
3900              { 0.6 \l_@@_xdots_shorten_end_dim }
3901          }
```

---

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

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

```
3902        \@@_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.

```
3903        \clist_if_empty:NF \l_@@_corners_clist
3904          {
3905            \bool_if:NTF \l_@@_no_cell_nodes_bool
3906              { \@@_error:n { corners~with~no-cell-nodes } }
3907              { \@@_compute_corners: }
3908          }
```

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

```
3909        \@@_adjust_pos_of_blocks_seq:
```

```
3910        \@@_deal_with_rounded_corners:
3911        \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3912        \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3913        \IfPackageLoadedT { tikz }
3914          {
3915            \tikzset
3916              {
3917                every~picture / .style =
3918                  {
3919                    overlay ,
3920                    remember~picture ,
3921                    name~prefix = \@@_env: -
3922                  }
3923              }
3924          }
3925        \bool_if:NT \c_@@_recent_array_bool
3926          { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3927        \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3928        \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3929        \cs_set_eq:NN \OverBrace \@@_OverBrace
3930        \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3931        \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3932        \cs_set_eq:NN \line \@@_line
3933        \g_@@_pre_code_after_tl
3934        \tl_gclear:N \g_@@_pre_code_after_tl
```

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

```
3935        \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).

```
3936        \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).

```
3937        \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3938          { \@@_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:`.

```
3939        \bool_set_true:N \l_@@_in_code_after_bool
3940        \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3941        \scan_stop:
3942        \tl_gclear:N \g_nicematrix_code_after_tl
3943        \group_end:
```

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

```
3944        \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3945        \tl_if_empty:NF \g_@@_pre_code_before_tl
3946          {
3947            \tl_gput_right:Ne \g_@@_aux_tl
3948              {
3949                \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3950                  { \exp_not:o \g_@@_pre_code_before_tl }
3951              }
3952            \tl_gclear:N \g_@@_pre_code_before_tl
3953          }
3954        \tl_if_empty:NF \g_nicematrix_code_before_tl
3955          {
3956            \tl_gput_right:Ne \g_@@_aux_tl
3957              {
3958                \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3959                  { \exp_not:o \g_nicematrix_code_before_tl }
3960              }
3961            \tl_gclear:N \g_nicematrix_code_before_tl
3962          }

3963        \str_gclear:N \g_@@_name_env_str
3964        \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[12]. 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.

```
3965        \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3966      }
```

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`.

```
3967    \NewDocumentCommand \@@_CodeAfter_keys: { O { } }
3968      { \keys_set:nn { nicematrix / CodeAfter } { #1 } }
```

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

```
3969    \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3970      {
```

---

[12]e.g. `\color[rgb]{0.5,0.5,0}`

```
3971      \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3972        { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3973    }
```

The following command must *not* be protected.

```
3974  \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3975    {
3976      { #1 }
3977      { #2 }
3978      {
3979        \int_compare:nNnTF { #3 } > { 98 }
3980          { \int_use:N \c@iRow }
3981          { #3 }
3982      }
3983      {
3984        \int_compare:nNnTF { #4 } > { 98 }
3985          { \int_use:N \c@jCol }
3986          { #4 }
3987      }
3988      { #5 }
3989    }
```

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).

```
3990  \hook_gput_code:nnn { begindocument } { . }
3991    {
3992      \cs_new_protected:Npe \@@_draw_dotted_lines:
3993        {
3994          \c_@@_pgfortikzpicture_tl
3995          \@@_draw_dotted_lines_i:
3996          \c_@@_endpgfortikzpicture_tl
3997        }
3998    }
```

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

```
3999  \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4000    {
4001      \pgfrememberpicturepositiononpagetrue
4002      \pgf@relevantforpicturesizefalse
4003      \g_@@_HVdotsfor_lines_tl
4004      \g_@@_Vdots_lines_tl
4005      \g_@@_Ddots_lines_tl
4006      \g_@@_Iddots_lines_tl
4007      \g_@@_Cdots_lines_tl
4008      \g_@@_Ldots_lines_tl
4009    }


4010  \cs_new_protected:Npn \@@_restore_iRow_jCol:
4011    {
4012      \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4013      \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4014    }
```

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

```
4015  \pgfdeclareshape { @@_diag_node }
4016    {
4017      \savedanchor { \five }
4018        {
4019          \dim_gset_eq:NN \pgf@x \l_tmpa_dim
```

99

```
4020        \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4021      }
4022    \anchor { 5 } { \five }
4023    \anchor { center } { \pgfpointorigin }
4024    \anchor { 1 }  { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4025    \anchor { 2 }  { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4026    \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4027    \anchor { 3 }  { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4028    \anchor { 4 }  { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4029    \anchor { 6 }  { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4030    \anchor { 7 }  { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4031    \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4032    \anchor { 8 }  { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4033    \anchor { 9 }  { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4034  }
```

The following command creates the diagonal nodes (in fact, if the matrix is not a square matrix, not all the nodes are on the diagonal).

```
4035 \cs_new_protected:Npn \@@_create_diag_nodes:
4036   {
4037     \pgfpicture
4038     \pgfrememberpicturepositiononpagetrue
4039     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4040       {
4041         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4042         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4043         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4044         \dim_set_eq:NN \l_tmpb_dim \pgf@y
4045         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4046         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4047         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4048         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4049         \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```
4050         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4051         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4052         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4053         \str_if_empty:NF \l_@@_name_str
4054           { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4055       }
```

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

```
4056     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4057     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4058     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4059     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4060     \pgfcoordinate
4061       { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4062     \pgfnodealias
4063       { \@@_env: - last }
4064       { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4065     \str_if_empty:NF \l_@@_name_str
4066       {
4067         \pgfnodealias
4068           { \l_@@_name_str - \int_use:N \l_tmpa_int }
4069           { \@@_env: - \int_use:N \l_tmpa_int }
4070         \pgfnodealias
4071           { \l_@@_name_str - last }
4072           { \@@_env: - last }
4073       }
```

```
4074        \endpgfpicture
4075    }
```

16   We draw the dotted lines

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

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

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

- the first argument is the row of the cell where the command was issued;

- the second argument is the column of the cell where the command was issued;

- the third argument is the x-value of the orientation vector of the line;

- the fourth argument is the y-value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;

- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;

- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```
4076  \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4077    {
```

First, we declare the current cell as "dotted" because we forbide intersections of dotted lines.

```
4078        \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4079        \int_set:Nn \l_@@_initial_i_int { #1 }
4080        \int_set:Nn \l_@@_initial_j_int { #2 }
4081        \int_set:Nn \l_@@_final_i_int { #1 }
4082        \int_set:Nn \l_@@_final_j_int { #2 }
```

We will do two loops: one when determinating the initial cell and the other when determinating 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.

```
4083        \bool_set_false:N \l_@@_stop_loop_bool
4084        \bool_do_until:Nn \l_@@_stop_loop_bool
4085          {
4086            \int_add:Nn \l_@@_final_i_int { #3 }
4087            \int_add:Nn \l_@@_final_j_int { #4 }
4088            \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.

```
4089          \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4090            \if_int_compare:w #3  = \c_one_int
4091              \bool_set_true:N \l_@@_final_open_bool
4092            \else:
4093              \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4094                \bool_set_true:N \l_@@_final_open_bool
4095              \fi:
4096            \fi:
4097          \else:
4098            \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4099              \if_int_compare:w #4 = -1
4100                \bool_set_true:N \l_@@_final_open_bool
4101              \fi:
4102            \else:
4103              \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4104                \if_int_compare:w #4 = \c_one_int
4105                  \bool_set_true:N \l_@@_final_open_bool
4106                \fi:
4107              \fi:
4108            \fi:
4109          \fi:

4110          \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.

```
4111              {
```

We do a step backwards.

```
4112              \int_sub:Nn \l_@@_final_i_int { #3 }
4113              \int_sub:Nn \l_@@_final_j_int { #4 }
4114              \bool_set_true:N \l_@@_stop_loop_bool
4115              }
```

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.

```
4116              {
4117              \cs_if_exist:cTF
4118                {
4119                  @@ _ dotted _
4120                  \int_use:N \l_@@_final_i_int -
4121                  \int_use:N \l_@@_final_j_int
4122                }
4123                {
4124                  \int_sub:Nn \l_@@_final_i_int { #3 }
4125                  \int_sub:Nn \l_@@_final_j_int { #4 }
4126                  \bool_set_true:N \l_@@_final_open_bool
4127                  \bool_set_true:N \l_@@_stop_loop_bool
4128                }
4129                {
4130                  \cs_if_exist:cTF
4131                    {
4132                      pgf @ sh @ ns @ \@@_env:
4133                      - \int_use:N \l_@@_final_i_int
4134                      - \int_use:N \l_@@_final_j_int
4135                    }
4136                    { \bool_set_true:N \l_@@_stop_loop_bool }
```

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

```
4137                      {
```

```
4138                     \cs_set_nopar:cpn
4139                       {
4140                         @@ _ dotted _
4141                         \int_use:N \l_@@_final_i_int -
4142                         \int_use:N \l_@@_final_j_int
4143                       }
4144                       { }
4145                   }
4146               }
4147           }
4148       }
```

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

```
4149       \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4150       \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }

4151       \bool_do_until:Nn \l_@@_stop_loop_bool
4152         {
4153           \int_sub:Nn \l_@@_initial_i_int { #3 }
4154           \int_sub:Nn \l_@@_initial_j_int { #4 }
4155           \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.

```
4156           \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4157             \if_int_compare:w #3 = \c_one_int
4158               \bool_set_true:N \l_@@_initial_open_bool
4159             \else:
```

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

```
4160               \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4161                 \bool_set_true:N \l_@@_initial_open_bool
4162               \fi:
4163             \fi:
4164           \else:
4165             \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4166               \if_int_compare:w #4 = \c_one_int
4167                 \bool_set_true:N \l_@@_initial_open_bool
4168               \fi:
4169             \else:
4170               \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4171                 \if_int_compare:w #4 = -1
4172                   \bool_set_true:N \l_@@_initial_open_bool
4173                 \fi:
4174               \fi:
4175             \fi:
4176           \fi:

4177           \bool_if:NTF \l_@@_initial_open_bool
4178             {
4179               \int_add:Nn \l_@@_initial_i_int { #3 }
4180               \int_add:Nn \l_@@_initial_j_int { #4 }
4181               \bool_set_true:N \l_@@_stop_loop_bool
4182             }
4183             {
4184               \cs_if_exist:cTF
4185                 {
4186                   @@ _ dotted _
4187                   \int_use:N \l_@@_initial_i_int -
4188                   \int_use:N \l_@@_initial_j_int
4189                 }
```

```
4190                    {
4191                      \int_add:Nn \l_@@_initial_i_int { #3 }
4192                      \int_add:Nn \l_@@_initial_j_int { #4 }
4193                      \bool_set_true:N \l_@@_initial_open_bool
4194                      \bool_set_true:N \l_@@_stop_loop_bool
4195                    }
4196                    {
4197                      \cs_if_exist:cTF
4198                        {
4199                          pgf @ sh @ ns @ \@@_env:
4200                          - \int_use:N \l_@@_initial_i_int
4201                          - \int_use:N \l_@@_initial_j_int
4202                        }
4203                        { \bool_set_true:N \l_@@_stop_loop_bool }
4204                        {
4205                          \cs_set_nopar:cpn
4206                            {
4207                              @@ _ dotted _
4208                              \int_use:N \l_@@_initial_i_int -
4209                              \int_use:N \l_@@_initial_j_int
4210                            }
4211                            { }
4212                        }
4213                    }
4214                }
4215          }
```

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.

```
4216        \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4217          {
4218            { \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.

```
4219            { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4220            { \int_use:N \l_@@_final_i_int }
4221            { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4222            { } % for the name of the block
4223          }
4224      }
```

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

```
4225  \cs_new_protected:Npn \@@_open_shorten:
4226    {
4227      \bool_if:NT \l_@@_initial_open_bool
4228        { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4229      \bool_if:NT \l_@@_final_open_bool
4230        { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4231    }
```

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

```
4232  \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4233    {
4234      \int_set_eq:NN \l_@@_row_min_int \c_one_int
```

```
4235    \int_set_eq:NN \l_@@_col_min_int \c_one_int
4236    \int_set_eq:NN \l_@@_row_max_int \c@iRow
4237    \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`.

```
4238    \seq_if_empty:NF \g_@@_submatrix_seq
4239      {
4240        \seq_map_inline:Nn \g_@@_submatrix_seq
4241          { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4242      }
4243  }
```

`#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 programmation of that command with the the standard syntax of L3.

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

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

```
4244  \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4245    {
4246      \if_int_compare:w #3 > #1
4247      \else:
4248        \if_int_compare:w #1 > #5
4249        \else:
4250          \if_int_compare:w #4 > #2
4251          \else:
4252            \if_int_compare:w #2 > #6
4253            \else:
4254              \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4255              \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4256              \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4257              \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4258            \fi:
4259          \fi:
4260        \fi:
4261      \fi:
4262    }
4263  \cs_new_protected:Npn \@@_set_initial_coords:
4264    {
4265      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4266      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4267    }
4268  \cs_new_protected:Npn \@@_set_final_coords:
4269    {
```

```
4270        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4271        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4272      }
4273    \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4274      {
4275        \pgfpointanchor
4276          {
4277            \@@_env:
4278            - \int_use:N \l_@@_initial_i_int
4279            - \int_use:N \l_@@_initial_j_int
4280          }
4281          { #1 }
4282        \@@_set_initial_coords:
4283      }
4284    \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4285      {
4286        \pgfpointanchor
4287          {
4288            \@@_env:
4289            - \int_use:N \l_@@_final_i_int
4290            - \int_use:N \l_@@_final_j_int
4291          }
4292          { #1 }
4293        \@@_set_final_coords:
4294      }

4295    \cs_new_protected:Npn \@@_open_x_initial_dim:
4296      {
4297        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4298        \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4299          {
4300            \cs_if_exist:cT
4301              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4302              {
4303                \pgfpointanchor
4304                  { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4305                  { west }
4306                \dim_set:Nn \l_@@_x_initial_dim
4307                  { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4308              }
4309          }
```

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

```
4310        \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4311          {
4312            \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4313            \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4314            \dim_add:Nn \l_@@_x_initial_dim \col@sep
4315          }
4316      }
4317    \cs_new_protected:Npn \@@_open_x_final_dim:
4318      {
4319        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4320        \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4321          {
4322            \cs_if_exist:cT
4323              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4324              {
4325                \pgfpointanchor
4326                  { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4327                  { east }
4328                \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4329                  { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4330              }
```

```
4331          }
```

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

```
4332     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4333       {
4334         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4335         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4336         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4337       }
4338   }
```

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.

```
4339 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4340   {
4341     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4342     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4343       {
4344         \@@_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.

```
4345         \group_begin:
4346           \@@_open_shorten:
4347           \int_if_zero:nTF { #1 }
4348             { \color { nicematrix-first-row } }
4349             {
```

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.

```
4350               \int_compare:nNnT { #1 } = \l_@@_last_row_int
4351                 { \color { nicematrix-last-row } }
4352             }
4353           \keys_set:nn { nicematrix / xdots } { #3 }
4354           \@@_color:o \l_@@_xdots_color_tl
4355           \@@_actually_draw_Ldots:
4356         \group_end:
4357       }
4358   }
```

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.

```
4359 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4360   {
4361     \bool_if:NTF \l_@@_initial_open_bool
4362       {
4363         \@@_open_x_initial_dim:
4364         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4365         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4366       }
4367       { \@@_set_initial_coords_from_anchor:n { base~east } }
```

```
4368      \bool_if:NTF \l_@@_final_open_bool
4369        {
4370          \@@_open_x_final_dim:
4371          \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4372          \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4373        }
4374        { \@@_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.

```
4375      \bool_lazy_all:nTF
4376        {
4377          \l_@@_initial_open_bool
4378          \l_@@_final_open_bool
4379          { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4380        }
4381        {
4382          \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4383          \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4384        }
```

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

```
4385        {
4386          \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4387          \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4388        }
4389      \@@_draw_line:
4390    }
```

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.

```
4391  \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4392    {
4393      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4394      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4395        {
4396          \@@_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.

```
4397          \group_begin:
4398            \@@_open_shorten:
4399            \int_if_zero:nTF { #1 }
4400              { \color { nicematrix-first-row } }
4401              {
```

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.

```
4402                \int_compare:nNnT { #1 } = \l_@@_last_row_int
4403                  { \color { nicematrix-last-row } }
4404              }
4405            \keys_set:nn { nicematrix / xdots } { #3 }
4406            \@@_color:o \l_@@_xdots_color_tl
4407            \@@_actually_draw_Cdots:
4408          \group_end:
4409        }
4410    }
```

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`.

```
4411 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4412   {
4413     \bool_if:NTF \l_@@_initial_open_bool
4414       { \@@_open_x_initial_dim: }
4415       { \@@_set_initial_coords_from_anchor:n { mid~east } }
4416     \bool_if:NTF \l_@@_final_open_bool
4417       { \@@_open_x_final_dim: }
4418       { \@@_set_final_coords_from_anchor:n { mid~west } }
4419     \bool_lazy_and:nnTF
4420       \l_@@_initial_open_bool
4421       \l_@@_final_open_bool
4422       {
4423         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4424         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4425         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4426         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4427         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4428       }
4429       {
4430         \bool_if:NT \l_@@_initial_open_bool
4431           { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4432         \bool_if:NT \l_@@_final_open_bool
4433           { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4434       }
4435     \@@_draw_line:
4436   }
4437 \cs_new_protected:Npn \@@_open_y_initial_dim:
4438   {
4439     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4440     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4441       {
4442         \cs_if_exist:cT
4443           { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4444           {
4445             \pgfpointanchor
4446               { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4447               { north }
4448             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4449               { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4450           }
4451       }
4452     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4453       {
4454         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4455         \dim_set:Nn \l_@@_y_initial_dim
4456           {
4457             \fp_to_dim:n
4458               {
4459                 \pgf@y
4460                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4461               }
4462           }
4463       }
4464   }
```

```
4465 \cs_new_protected:Npn \@@_open_y_final_dim:
4466   {
4467     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4468     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4469       {
4470         \cs_if_exist:cT
4471           { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4472           {
4473             \pgfpointanchor
4474               { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4475               { south }
4476             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4477               { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4478           }
4479       }
4480     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4481       {
4482         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4483         \dim_set:Nn \l_@@_y_final_dim
4484           { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } } }
4485       }
4486   }
```

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.

```
4487 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4488   {
4489     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4490     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4491       {
4492         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 0
```

The previous command may have changed the current environment by marking some cells as "dotted", but, fortunately, it is outside the group for the options of the line.

```
4493         \group_begin:
4494           \@@_open_shorten:
4495           \int_if_zero:nTF { #2 }
4496             { \color { nicematrix-first-col } }
4497             {
4498               \int_compare:nNnT { #2 } = \l_@@_last_col_int
4499                 { \color { nicematrix-last-col } }
4500             }
4501           \keys_set:nn { nicematrix / xdots } { #3 }
4502           \@@_color:o \l_@@_xdots_color_tl
4503           \@@_actually_draw_Vdots:
4504         \group_end:
4505       }
4506   }
```

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`.

```
4507 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4508   {
```

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

```
4509          \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.

```
4510            {
4511              \@@_open_y_initial_dim:
4512              \@@_open_y_final_dim:
4513              \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4514              {
4515                \@@_qpoint:n { col - 1 }
4516                \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4517                \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4518                \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4519                \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4520              }
4521              {
4522                \bool_lazy_and:nnTF
4523                  { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4524                  { \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".

```
4525                {
4526                  \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4527                  \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4528                  \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4529                  \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4530                  \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4531                }
```

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

```
4532                {
4533                  \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4534                  \dim_set_eq:NN \l_tmpa_dim \pgf@x
4535                  \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4536                  \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4537                }
4538              }
4539          }
```

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.

```
4540          {
4541            \bool_set_false:N \l_tmpa_bool
4542            \bool_if:NF \l_@@_initial_open_bool
4543              {
4544                \bool_if:NF \l_@@_final_open_bool
4545                  {
4546                    \@@_set_initial_coords_from_anchor:n { south~west }
4547                    \@@_set_final_coords_from_anchor:n { north~west }
4548                    \bool_set:Nn \l_tmpa_bool
4549                      { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4550                  }
4551              }
```

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

```
4552            \bool_if:NTF \l_@@_initial_open_bool
4553              {
4554                \@@_open_y_initial_dim:
4555                \@@_set_final_coords_from_anchor:n { north }
4556                \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4557              }
4558              {
4559                \@@_set_initial_coords_from_anchor:n { south }
4560                \bool_if:NTF \l_@@_final_open_bool
```

```
4561                \@@_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.

```
4562                {
4563                  \@@_set_final_coords_from_anchor:n { north }
4564                  \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4565                    {
4566                      \dim_set:Nn \l_@@_x_initial_dim
4567                        {
4568                          \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4569                            \l_@@_x_initial_dim \l_@@_x_final_dim
4570                        }
4571                    }
4572                }
4573            }
4574        }
4575      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4576      \@@_draw_line:
4577    }
```

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.

```
4578  \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4579    {
4580      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4581      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4582        {
4583          \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 1
```

The previous command may have changed the current environment by marking some cells as "dotted", but, fortunately, it is outside the group for the options of the line.

```
4584          \group_begin:
4585            \@@_open_shorten:
4586            \keys_set:nn { nicematrix / xdots } { #3 }
4587            \@@_color:o \l_@@_xdots_color_tl
4588            \@@_actually_draw_Ddots:
4589          \group_end:
4590        }
4591    }
```

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`.

```
4592  \cs_new_protected:Npn \@@_actually_draw_Ddots:
4593    {
4594      \bool_if:NTF \l_@@_initial_open_bool
4595        {
4596          \@@_open_y_initial_dim:
4597          \@@_open_x_initial_dim:
```

```
4598          }
4599          { \@@_set_initial_coords_from_anchor:n { south~east } }
4600        \bool_if:NTF \l_@@_final_open_bool
4601          {
4602            \@@_open_x_final_dim:
4603            \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4604          }
4605          { \@@_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.

```
4606        \bool_if:NT \l_@@_parallelize_diags_bool
4607          {
4608            \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).

```
4609            \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.

```
4610            {
4611              \dim_gset:Nn \g_@@_delta_x_one_dim
4612                { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4613              \dim_gset:Nn \g_@@_delta_y_one_dim
4614                { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4615            }
```

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`.

```
4616            {
4617              \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4618                {
4619                  \dim_set:Nn \l_@@_y_final_dim
4620                    {
4621                      \l_@@_y_initial_dim +
4622                      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4623                      \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4624                    }
4625                }
4626            }
4627          }
4628        \@@_draw_line:
4629      }
```

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.

```
4630  \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4631    {
4632      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4633      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4634        {
4635          \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }
```

The previous command may have changed the current environment by marking some cells as "dotted", but, fortunately, it is outside the group for the options of the line.

```
4636          \group_begin:
4637            \@@_open_shorten:
4638            \keys_set:nn { nicematrix / xdots } { #3 }
4639            \@@_color:o \l_@@_xdots_color_tl
4640            \@@_actually_draw_Iddots:
4641          \group_end:
4642        }
4643    }
```

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.`

```
4644 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4645   {
4646     \bool_if:NTF \l_@@_initial_open_bool
4647       {
4648         \@@_open_y_initial_dim:
4649         \@@_open_x_initial_dim:
4650       }
4651       { \@@_set_initial_coords_from_anchor:n { south~west } }
4652     \bool_if:NTF \l_@@_final_open_bool
4653       {
4654         \@@_open_y_final_dim:
4655         \@@_open_x_final_dim:
4656       }
4657       { \@@_set_final_coords_from_anchor:n { north~east } }
4658     \bool_if:NT \l_@@_parallelize_diags_bool
4659       {
4660         \int_gincr:N \g_@@_iddots_int
4661         \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4662           {
4663             \dim_gset:Nn \g_@@_delta_x_two_dim
4664               { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4665             \dim_gset:Nn \g_@@_delta_y_two_dim
4666               { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4667           }
4668           {
4669             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4670               {
4671                 \dim_set:Nn \l_@@_y_final_dim
4672                   {
4673                     \l_@@_y_initial_dim +
4674                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4675                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4676                   }
4677               }
4678           }
4679       }
4680     \@@_draw_line:
4681   }
```

17   The actual instructions for drawing the dotted lines with Tikz

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

- `\l_@@_x_initial_dim`

- `\l_@@_y_initial_dim`

- `\l_@@_x_final_dim`

- `\l_@@_y_final_dim`

- `\l_@@_initial_open_bool`

- `\l_@@_final_open_bool`

```
4682 \cs_new_protected:Npn \@@_draw_line:
4683   {
4684     \pgfremberpicturepositiononpagetrue
4685     \pgf@relevantforpicturesizefalse
4686     \bool_lazy_or:nnTF
4687       { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4688     \l_@@_dotted_bool
4689     \@@_draw_standard_dotted_line:
4690     \@@_draw_unstandard_dotted_line:
4691   }
```

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.

```
4692 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4693   {
4694     \begin { scope }
4695     \@@_draw_unstandard_dotted_line:o
4696       { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4697   }
```

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 diredtly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```
4698 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4699 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4700   {
4701     \@@_draw_unstandard_dotted_line:nooo
4702       { #1 }
4703     \l_@@_xdots_up_tl
4704     \l_@@_xdots_down_tl
4705     \l_@@_xdots_middle_tl
4706   }
```

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

```
4707 \hook_gput_code:nnn { begindocument } { . }
4708   {
4709     \IfPackageLoadedT { tikz }
4710       {
4711         \tikzset
4712           {
4713             @@_node_above / .style = { sloped , above } ,
4714             @@_node_below / .style = { sloped , below } ,
4715             @@_node_middle / .style =
4716               {
4717                 sloped ,
4718                 inner~sep = \c_@@_innersep_middle_dim
4719               }
4720           }
4721       }
4722   }
```

```
4723  \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4724  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4725    {
```

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.

```
4726        \dim_zero_new:N \l_@@_l_dim
4727        \dim_set:Nn \l_@@_l_dim
4728          {
4729            \fp_to_dim:n
4730              {
4731                sqrt
4732                  (
4733                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4734                      +
4735                    ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4736                  )
4737              }
4738          }
```

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.

```
4739        \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4740          {
4741            \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4742              \@@_draw_unstandard_dotted_line_i:
4743          }
```

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

```
4744        \bool_if:NT \l_@@_xdots_h_labels_bool
4745          {
4746            \tikzset
4747              {
4748                @@_node_above / .style = { auto = left } ,
4749                @@_node_below / .style = { auto = right } ,
4750                @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4751              }
4752          }
4753        \tl_if_empty:nF { #4 }
4754          { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4755        \draw
4756          [ #1 ]
4757            ( \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).

```
4758            -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4759              node [ @@_node_below ] { $ \scriptstyle #3 $ }
4760              node [ @@_node_above ] { $ \scriptstyle #2 $ }
4761              ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4762        \end { scope }
4763    }
4764  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4765    {
4766      \dim_set:Nn \l_tmpa_dim
4767        {
4768          \l_@@_x_initial_dim
4769          + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
```

116

```
4770          * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4771        }
4772      \dim_set:Nn \l_tmpb_dim
4773        {
4774          \l_@@_y_initial_dim
4775          + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4776          * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4777        }
4778      \dim_set:Nn \l_@@_tmpc_dim
4779        {
4780          \l_@@_x_final_dim
4781          - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4782          * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4783        }
4784      \dim_set:Nn \l_@@_tmpd_dim
4785        {
4786          \l_@@_y_final_dim
4787          - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4788          * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4789        }
4790      \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4791      \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4792      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4793      \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4794    }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```
4795 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4796    {
4797      \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.

```
4798      \dim_zero_new:N \l_@@_l_dim
4799      \dim_set:Nn \l_@@_l_dim
4800        {
4801          \fp_to_dim:n
4802            {
4803              sqrt
4804              (
4805                ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4806                  +
4807                ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4808              )
4809            }
4810        }
```

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.

```
4811      \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4812        {
4813          \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4814            \@@_draw_standard_dotted_line_i:
4815        }
4816      \group_end:
4817      \bool_lazy_all:nF
4818        {
4819          { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4820          { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4821          { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
```

```
4822          }
4823        \l_@@_labels_standard_dotted_line:
4824      }

4825    \dim_const:Nn \c_@@_max_l_dim { 50 cm }

4826    \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4827      {
```

The number of dots will be `\l_tmpa_int + 1`.

```
4828        \int_set:Nn \l_tmpa_int
4829          {
4830            \dim_ratio:nn
4831              {
4832                \l_@@_l_dim
4833                - \l_@@_xdots_shorten_start_dim
4834                - \l_@@_xdots_shorten_end_dim
4835              }
4836              \l_@@_xdots_inter_dim
4837          }
```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```
4838        \dim_set:Nn \l_tmpa_dim
4839          {
4840            ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4841            \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4842          }
4843        \dim_set:Nn \l_tmpb_dim
4844          {
4845            ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4846            \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4847          }
```

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.

```
4848        \dim_gadd:Nn \l_@@_x_initial_dim
4849          {
4850            ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4851            \dim_ratio:nn
4852              {
4853                \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4854                + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4855              }
4856              { 2 \l_@@_l_dim }
4857          }
4858        \dim_gadd:Nn \l_@@_y_initial_dim
4859          {
4860            ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4861            \dim_ratio:nn
4862              {
4863                \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4864                + \l_@@_xdots_shorten_start_dim  - \l_@@_xdots_shorten_end_dim
4865              }
4866              { 2 \l_@@_l_dim }
4867          }
4868        \pgf@relevantforpicturesizefalse
4869        \int_step_inline:nnn \c_zero_int \l_tmpa_int
4870          {
4871            \pgfpathcircle
4872              { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4873              { \l_@@_xdots_radius_dim }
4874            \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4875            \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4876          }
```

```
4877        \pgfusepathqfill
4878    }


4879  \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4880    {
4881      \pgfscope
4882      \pgftransformshift
4883        {
4884          \pgfpointlineattime { 0.5 }
4885            { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4886            { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4887        }
4888      \fp_set:Nn \l_tmpa_fp
4889        {
4890          atand
4891          (
4892            \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4893            \l_@@_x_final_dim - \l_@@_x_initial_dim
4894          )
4895        }
4896      \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4897      \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4898      \tl_if_empty:NF \l_@@_xdots_middle_tl
4899        {
4900          \begin { pgfscope }
4901          \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4902          \pgfnode
4903            { rectangle }
4904            { center }
4905            {
4906              \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4907                {
4908                  \c_math_toggle_token
4909                  \scriptstyle \l_@@_xdots_middle_tl
4910                  \c_math_toggle_token
4911                }
4912            }
4913            { }
4914            {
4915              \pgfsetfillcolor { white }
4916              \pgfusepath { fill }
4917            }
4918          \end { pgfscope }
4919        }
4920      \tl_if_empty:NF \l_@@_xdots_up_tl
4921        {
4922          \pgfnode
4923            { rectangle }
4924            { south }
4925            {
4926              \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4927                {
4928                  \c_math_toggle_token
4929                  \scriptstyle \l_@@_xdots_up_tl
4930                  \c_math_toggle_token
4931                }
4932            }
4933            { }
4934            { \pgfusepath { } }
4935        }
4936      \tl_if_empty:NF \l_@@_xdots_down_tl
4937        {
4938          \pgfnode
```

```
4939            { rectangle }
4940            { north }
4941            {
4942              \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4943                {
4944                  \c_math_toggle_token
4945                  \scriptstyle \l_@@_xdots_down_tl
4946                  \c_math_toggle_token
4947                }
4948            }
4949            { }
4950            { \pgfusepath { } }
4951        }
4952      \endpgfscope
4953    }
```

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 thats' 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.

```
4954  \hook_gput_code:nnn { begindocument } { . }
4955    {
4956      \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4957      \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4958      \cs_new_protected:Npn \@@_Ldots
4959        { \@@_collect_options:n { \@@_Ldots_i } }
4960      \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4961        {
4962          \int_if_zero:nTF \c@jCol
4963            { \@@_error:nn { in~first~col } \Ldots }
4964            {
4965              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4966                { \@@_error:nn { in~last~col } \Ldots }
4967                {
4968                  \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4969                    { #1 , down = #2 , up = #3 , middle = #4 }
4970                }
4971            }
4972          \bool_if:NF \l_@@_nullify_dots_bool
4973            { \phantom { \ensuremath { \@@_old_ldots } } }
4974          \bool_gset_true:N \g_@@_empty_cell_bool
4975        }


4976      \cs_new_protected:Npn \@@_Cdots
4977        { \@@_collect_options:n { \@@_Cdots_i } }
4978      \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4979        {
4980          \int_if_zero:nTF \c@jCol
4981            { \@@_error:nn { in~first~col } \Cdots }
4982            {
4983              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
```

```
4984                { \@@_error:nn { in~last~col } \Cdots }
4985                {
4986                  \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4987                    { #1 , down = #2 , up = #3 , middle = #4 }
4988                }
4989            }
4990          \bool_if:NF \l_@@_nullify_dots_bool
4991            { \phantom { \ensuremath { \@@_old_cdots } } }
4992          \bool_gset_true:N \g_@@_empty_cell_bool
4993        }


4994      \cs_new_protected:Npn \@@_Vdots
4995        { \@@_collect_options:n { \@@_Vdots_i } }
4996      \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4997        {
4998          \int_if_zero:nTF \c@iRow
4999            { \@@_error:nn { in~first~row } \Vdots }
5000            {
5001              \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5002                { \@@_error:nn { in~last~row } \Vdots }
5003                {
5004                  \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5005                    { #1 , down = #2 , up = #3 , middle = #4 }
5006                }
5007            }
5008          \bool_if:NF \l_@@_nullify_dots_bool
5009            { \phantom { \ensuremath { \@@_old_vdots } } }
5010          \bool_gset_true:N \g_@@_empty_cell_bool
5011        }


5012      \cs_new_protected:Npn \@@_Ddots
5013        { \@@_collect_options:n { \@@_Ddots_i } }
5014      \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5015        {
5016          \int_case:nnF \c@iRow
5017            {
5018              0                  { \@@_error:nn { in~first~row } \Ddots }
5019              \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5020            }
5021            {
5022              \int_case:nnF \c@jCol
5023                {
5024                  0                  { \@@_error:nn { in~first~col } \Ddots }
5025                  \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5026                }
5027                {
5028                  \keys_set_known:nn { nicematrix / Ddots } { #1 }
5029                  \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5030                    { #1 , down = #2 , up = #3 , middle = #4 }
5031                }

5033            }
5034          \bool_if:NF \l_@@_nullify_dots_bool
5035            { \phantom { \ensuremath { \@@_old_ddots } } }
5036          \bool_gset_true:N \g_@@_empty_cell_bool
5037        }


5038      \cs_new_protected:Npn \@@_Iddots
5039        { \@@_collect_options:n { \@@_Iddots_i } }
5040      \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5041        {
```

```
5042        \int_case:nnF \c@iRow
5043          {
5044            0                    { \@@_error:nn { in~first~row } \Iddots }
5045            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5046          }
5047          {
5048            \int_case:nnF \c@jCol
5049              {
5050                0                    { \@@_error:nn { in~first~col } \Iddots }
5051                \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5052              }
5053              {
5054                \keys_set_known:nn { nicematrix / Ddots } { #1 }
5055                \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5056                  { #1 , down = #2 , up = #3 , middle = #4 }
5057              }
5058          }
5059        \bool_if:NF \l_@@_nullify_dots_bool
5060          { \phantom { \ensuremath { \@@_old_iddots } } } }
5061        \bool_gset_true:N \g_@@_empty_cell_bool
5062      }
5063  }
```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```
5064 \keys_define:nn { nicematrix / Ddots }
5065   {
5066     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5067     draw-first .default:n = true ,
5068     draw-first .value_forbidden:n = true
5069   }
```

The command `\@@_Hspace:` will be linked to `\hspace` in {NiceArray}.

```
5070 \cs_new_protected:Npn \@@_Hspace:
5071   {
5072    \bool_gset_true:N \g_@@_empty_cell_bool
5073    \hspace
5074   }
```

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`.

```
5075 \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`.

```
5076 \cs_new:Npn \@@_Hdotsfor:
5077   {
5078     \bool_lazy_and:nnTF
5079       { \int_if_zero_p:n \c@jCol }
5080       { \int_if_zero_p:n \l_@@_first_col_int }
5081       {
5082         \bool_if:NTF \g_@@_after_col_zero_bool
5083           {
5084             \multicolumn { 1 } { c } { }
5085             \@@_Hdotsfor_i
5086           }
5087           { \@@_fatal:n { Hdotsfor~in~col~0 } }
5088       }
5089       {
```

```
5090        \multicolumn { 1 } { c } { }
5091        \@@_Hdotsfor_i
5092      }
5093  }
```

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:`).

```
5094  \hook_gput_code:nnn { begindocument } { . }
5095    {
5096      \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5097      \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.

```
5098      \cs_new_protected:Npn \@@_Hdotsfor_i
5099        { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5100      \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5101        {
5102          \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5103            {
5104              \@@_Hdotsfor:nnnn
5105                { \int_use:N \c@iRow }
5106                { \int_use:N \c@jCol }
5107                { #2 }
5108                {
5109                  #1 , #3 ,
5110                  down = \exp_not:n { #4 } ,
5111                  up = \exp_not:n { #5 } ,
5112                  middle = \exp_not:n { #6 }
5113                }
5114            }
5115          \prg_replicate:nn { #2 - 1 }
5116            {
5117              &
5118              \multicolumn { 1 } { c } { }
5119              \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5120            }
5121        }
5122    }
```

```
5123  \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5124    {
5125      \bool_set_false:N \l_@@_initial_open_bool
5126      \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5127      \int_set:Nn \l_@@_initial_i_int { #1 }
5128      \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
5129      \int_compare:nNnTF { #2 } = \c_one_int
5130        {
5131          \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5132          \bool_set_true:N \l_@@_initial_open_bool
5133        }
5134        {
5135          \cs_if_exist:cTF
5136            {
5137              pgf @ sh @ ns @ \@@_env:
5138              - \int_use:N \l_@@_initial_i_int
5139              - \int_eval:n { #2 - 1 }
5140            }
5141            { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5142            {
```

```
5143            \int_set:Nn \l_@@_initial_j_int { #2 }
5144            \bool_set_true:N \l_@@_initial_open_bool
5145          }
5146      }
5147    \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
5148      {
5149        \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5150        \bool_set_true:N \l_@@_final_open_bool
5151      }
5152      {
5153        \cs_if_exist:cTF
5154          {
5155            pgf @ sh @ ns @ \@@_env:
5156            - \int_use:N \l_@@_final_i_int
5157            - \int_eval:n { #2 + #3 }
5158          }
5159          { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5160          {
5161            \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5162            \bool_set_true:N \l_@@_final_open_bool
5163          }
5164      }
5165    \group_begin:
5166    \@@_open_shorten:
5167    \int_if_zero:nTF { #1 }
5168      { \color { nicematrix-first-row } }
5169      {
5170        \int_compare:nNnT { #1 } = \g_@@_row_total_int
5171          { \color { nicematrix-last-row } }
5172      }
5173
5174    \keys_set:nn { nicematrix / xdots } { #4 }
5175    \@@_color:o \l_@@_xdots_color_tl
5176    \@@_actually_draw_Ldots:
5177    \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).

```
5178        \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5179          { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5180    }


5181 \hook_gput_code:nnn { begindocument } { . }
5182   {
5183     \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5184     \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5185     \cs_new_protected:Npn \@@_Vdotsfor:
5186       { \@@_collect_options:n { \@@_Vdotsfor_i } }
5187     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5188       {
5189         \bool_gset_true:N \g_@@_empty_cell_bool
5190         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5191           {
5192             \@@_Vdotsfor:nnnn
5193               { \int_use:N \c@iRow }
5194               { \int_use:N \c@jCol }
5195               { #2 }
5196               {
5197                 #1 , #3 ,
5198                 down = \exp_not:n { #4 } ,
5199                 up = \exp_not:n { #5 } ,
```

124

```
5200                 middle = \exp_not:n { #6 }
5201               }
5202           }
5203         }
5204     }


5205 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5206   {
5207     \bool_set_false:N \l_@@_initial_open_bool
5208     \bool_set_false:N \l_@@_final_open_bool
```
For the column, it's easy.
```
5209     \int_set:Nn \l_@@_initial_j_int { #2 }
5210     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```
For the row, it's a bit more complicated.
```
5211     \int_compare:nNnTF { #1 } = \c_one_int
5212       {
5213         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5214         \bool_set_true:N \l_@@_initial_open_bool
5215       }
5216       {
5217         \cs_if_exist:cTF
5218           {
5219             pgf @ sh @ ns @ \@@_env:
5220             - \int_eval:n { #1 - 1 }
5221             - \int_use:N \l_@@_initial_j_int
5222           }
5223           { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5224           {
5225             \int_set:Nn \l_@@_initial_i_int { #1 }
5226             \bool_set_true:N \l_@@_initial_open_bool
5227           }
5228       }
5229     \int_compare:nNnTF { #1 + #3 -1 } = \c@iRow
5230       {
5231         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5232         \bool_set_true:N \l_@@_final_open_bool
5233       }
5234       {
5235         \cs_if_exist:cTF
5236           {
5237             pgf @ sh @ ns @ \@@_env:
5238             - \int_eval:n { #1 + #3 }
5239             - \int_use:N \l_@@_final_j_int
5240           }
5241           { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5242           {
5243             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5244             \bool_set_true:N \l_@@_final_open_bool
5245           }
5246       }
5247     \group_begin:
5248     \@@_open_shorten:
5249     \int_if_zero:nTF { #2 }
5250       { \color { nicematrix-first-col } }
5251       {
5252         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5253           { \color { nicematrix-last-col } }
5254       }
5255     \keys_set:nn { nicematrix / xdots } { #4 }
5256     \@@_color:o \l_@@_xdots_color_tl
5257     \@@_actually_draw_Vdots:
5258     \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).

```
5259      \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5260        { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5261    }
```

The command \@@_rotate: will be linked to \rotate in {NiceArrayWithDelims}.

```
5262  \NewDocumentCommand \@@_rotate: { O { } }
5263    {
5264      \peek_remove_spaces:n
5265        {
5266          \bool_gset_true:N \g_@@_rotate_bool
5267          \keys_set:nn { nicematrix / rotate } { #1 }
5268        }
5269    }
```

```
5270  \keys_define:nn { nicematrix / rotate }
5271    {
5272      c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5273      c .value_forbidden:n = true ,
5274      unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5275    }
```

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, if 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]

```
5276  \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5277    {
5278      \tl_if_empty:nTF { #2 }
5279        { #1 }
5280        { \@@_double_int_eval_i:n #1-#2 \q_stop }
5281    }
5282  \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5283    { \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).

```
5284  \hook_gput_code:nnn { begindocument } { . }
5285    {
```

---

[13]Indeed, we want that the user may use the command \line in \CodeAfter with LaTeX counters in the arguments — with the command \value.

126

```
5286    \cs_set_nopar:Npn \l_@@_argspec_tl
5287      { O { } m m ! O { } E { _ ^ : } { { } { } { } } }
5288    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5289    \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5290      {
5291        \group_begin:
5292        \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5293        \@@_color:o \l_@@_xdots_color_tl
5294        \use:e
5295          {
5296            \@@_line_i:nn
5297              { \@@_double_int_eval:n #2 - \q_stop }
5298              { \@@_double_int_eval:n #3 - \q_stop }
5299          }
5300        \group_end:
5301      }
5302  }
5303  \cs_new_protected:Npn \@@_line_i:nn #1 #2
5304    {
5305      \bool_set_false:N \l_@@_initial_open_bool
5306      \bool_set_false:N \l_@@_final_open_bool
5307      \bool_lazy_or:nnTF
5308        { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5309        { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5310        { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```
5311        { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5312    }
5313  \hook_gput_code:nnn { begindocument } { . }
5314    {
5315      \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5316        {
```

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:`.

```
5317          \c_@@_pgfortikzpicture_tl
5318          \@@_draw_line_iii:nn { #1 } { #2 }
5319          \c_@@_endpgfortikzpicture_tl
5320        }
5321    }
```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```
5322  \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5323    {
5324      \pgfrememberpicturepositiononpagetrue
5325      \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5326      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5327      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5328      \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5329      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5330      \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5331      \@@_draw_line:
5332    }
```

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

```
5333 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5334   { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

5335 \cs_new:Npn \@@_if_col_greater_than:nn #1 #2
5336   { \int_compare:nNnF { \c@jCol } < { #1 } { #2 }  }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5337 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5338 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5339   {
5340     \tl_gput_right:Ne \g_@@_row_style_tl
5341       {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5342         \exp_not:N
5343         \@@_if_row_less_than:nn
5344           { \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`).

```
5345           {
5346             \exp_not:N
5347             \@@_if_col_greater_than:nn
5348               { \int_eval:n { \c@jCol } }
5349               { \exp_not:n { #1 } \scan_stop: }
5350           }
5351       }
5352   }


5353 \keys_define:nn { nicematrix / RowStyle }
5354   {
5355     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5356     cell-space-top-limit .value_required:n = true ,
5357     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5358     cell-space-bottom-limit .value_required:n = true ,
5359     cell-space-limits .meta:n =
5360       {
5361         cell-space-top-limit = #1 ,
5362         cell-space-bottom-limit = #1 ,
5363       } ,
5364     color .tl_set:N = \l_@@_color_tl ,
5365     color .value_required:n = true ,
5366     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5367     bold .default:n = true ,
5368     nb-rows .code:n =
5369       \str_if_eq:eeTF { #1 } { * }
5370         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5371         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
```

```
5372        nb-rows .value_required:n = true ,
5373        fill .tl_set:N = \l_@@_fill_tl ,
5374        fill .value_required:n = true ,
5375        opacity .tl_set:N = \l_@@_opacity_tl ,
5376        opacity .value_required:n = true ,
5377        rowcolor .tl_set:N = \l_@@_fill_tl ,
5378        rowcolor .value_required:n = true ,
5379        rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5380        rounded-corners .default:n = 4 pt ,
5381        unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5382    }


5383 \NewDocumentCommand \@@_RowStyle:n { O { } m }
5384    {
5385      \group_begin:
5386      \tl_clear:N \l_@@_fill_tl
5387      \tl_clear:N \l_@@_opacity_tl
5388      \tl_clear:N \l_@@_color_tl
5389      \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5390      \dim_zero:N \l_@@_rounded_corners_dim
5391      \dim_zero:N \l_tmpa_dim
5392      \dim_zero:N \l_tmpb_dim
5393      \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key `rowcolor` (of its alias `fill`) has been used.

```
5394      \tl_if_empty:NF \l_@@_fill_tl
5395        {
5396          \@@_add_opacity_to_fill:
5397          \tl_gput_right:Ne \g_@@_pre_code_before_tl
5398            {
```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the commande applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```
5399            \int_compare:nNnTF \c@jCol > \c_one_int
5400              {
```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row). The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5401            \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5402              { \int_use:N \c@iRow - \int_use:N \c@jCol }
5403              { \int_use:N \c@iRow - * }
5404              { \dim_use:N \l_@@_rounded_corners_dim }
```

Then, the other rows (if there are several rows).

```
5405            \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5406              { \@@_rounded_from_row:n { \c@iRow + 1 } }
5407              }
```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```
5408              { \@@_rounded_from_row:n { \c@iRow } }
5409            }
5410        }
5411      \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```
5412      \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5413        {
5414          \@@_put_in_row_style:e
5415            {
5416              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5417                {
```

It's not possible to chanage the following code by using \dim_set_eq:NN (because of expansion).

```
5418            \dim_set:Nn \l_@@_cell_space_top_limit_dim
5419              { \dim_use:N \l_tmpa_dim }
5420          }
5421        }
5422      }
```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```
5423      \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5424        {
5425          \@@_put_in_row_style:e
5426            {
5427              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5428                {
5429                  \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5430                    { \dim_use:N \l_tmpb_dim }
5431                }
5432            }
5433        }
```

\l_@@_color_tl is the value of the key color of \RowStyle.

```
5434      \tl_if_empty:NF \l_@@_color_tl
5435        {
5436          \@@_put_in_row_style:e
5437            {
5438              \mode_leave_vertical:
5439              \@@_color:n { \l_@@_color_tl }
5440            }
5441        }
```

\l_@@_bold_row_style_bool is the value of the key bold.

```
5442      \bool_if:NT \l_@@_bold_row_style_bool
5443        {
5444          \@@_put_in_row_style:n
5445            {
5446              \exp_not:n
5447                {
5448                  \if_mode_math:
5449                    \c_math_toggle_token
5450                    \bfseries \boldmath
5451                    \c_math_toggle_token
5452                  \else:
5453                    \bfseries \boldmath
5454                  \fi:
5455                }
5456            }
5457        }
5458      \group_end:
5459      \g_@@_row_style_tl
5460      \ignorespaces
5461    }
```

The following commande must *not* be protected.

```
5462 \cs_new:Npn \@@_rounded_from_row:n #1
5463   {
5464     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
```

In the following code, the "- 1" is *not* a subtraction.

```
5465        { \int_eval:n { #1 } - 1 }
5466        {
5467          \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5468          - \exp_not:n { \int_use:N \c@jCol }
5469        }
5470        { \dim_use:N \l_@@_rounded_corners_dim }
5471    }
```

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).

```
5472 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5473 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5474 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5475   {
```

Firt, 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.

```
5476     \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.

```
5477     \str_if_in:nnF { #1 } { !! }
5478       {
5479         \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use \str_if_eq:eeTF which is slightly faster than \tl_if_eq:nnTF.

```
5480           { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } } }
5481       }
5482     \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5483       {
5484         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5485         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5486       }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position \l_tmpa_int).

```
5487       { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5488   }
```

The following command must be used within a \pgfpicture.

```
5489 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5490   {
5491     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5492       {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5493        \group_begin:
5494        \pgfsetcornersarced
5495          {
5496            \pgfpoint
5497              { \l_@@_tab_rounded_corners_dim }
5498              { \l_@@_tab_rounded_corners_dim }
5499          }
```

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.

```
5500        \bool_if:NTF \l_@@_hvlines_bool
5501          {
5502            \pgfpathrectanglecorners
5503              {
5504                \pgfpointadd
5505                  { \@@_qpoint:n { row-1 } }
5506                  { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5507              }
5508              {
5509                \pgfpointadd
5510                  {
5511                    \@@_qpoint:n
5512                      { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5513                  }
5514                  { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5515              }
5516          }
5517          {
5518            \pgfpathrectanglecorners
5519              { \@@_qpoint:n { row-1 } }
5520              {
5521                \pgfpointadd
5522                  {
5523                    \@@_qpoint:n
5524                      { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5525                  }
5526                  { \pgfpoint \c_zero_dim \arrayrulewidth }
5527              }
5528          }
5529        \pgfusepath { clip }
5530        \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```
5531        }
5532    }
```

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`).

```
5533 \cs_new_protected:Npn \@@_actually_color:
5534    {
5535      \pgfpicture
5536      \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.

```
5537      \@@_clip_with_rounded_corners:
5538      \seq_map_indexed_inline:Nn \g_@@_colors_seq
5539        {
5540          \int_compare:nNnTF { ##1 } = \c_one_int
```

132

```
5541              {
5542                \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5543                \use:c { g_@@_color _ 1 _tl }
5544                \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5545              }
5546              {
5547                \begin { pgfscope }
5548                  \@@_color_opacity ##2
5549                  \use:c { g_@@_color _ ##1 _tl }
5550                  \tl_gclear:c { g_@@_color _ ##1 _tl }
5551                  \pgfusepath { fill }
5552                \end { pgfscope }
5553              }
5554          }
5555        \endpgfpicture
5556    }
```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```
5557  \cs_new_protected:Npn \@@_color_opacity
5558    {
5559      \peek_meaning:NTF [
5560        { \@@_color_opacity:w }
5561        { \@@_color_opacity:w [ ] }
5562    }
```

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.

```
5563  \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5564    {
5565      \tl_clear:N \l_tmpa_tl
5566      \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.

```
5567      \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5568      \tl_if_empty:NTF \l_tmpb_tl
5569        { \@declaredcolor }
5570        { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } } }
5571    }
```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```
5572  \keys_define:nn { nicematrix / color-opacity }
5573    {
5574      opacity .tl_set:N         = \l_tmpa_tl ,
5575      opacity .value_required:n = true
5576    }
```

```
5577  \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5578    {
5579      \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5580      \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5581      \@@_cartesian_path:
5582    }
```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```
5583  \NewDocumentCommand \@@_rowcolor { O { } m m }
5584    {
5585      \tl_if_blank:nF { #2 }
5586        {
```

133

```
5587        \@@_add_to_colors_seq:en
5588          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5589          { \@@_cartesian_color:nn { #3 } { - } }
5590      }
5591    }
```

Here an example : \@@_columncolor:nn {red!15} {1,3,5-7,10-}

```
5592  \NewDocumentCommand \@@_columncolor { O { } m m }
5593    {
5594      \tl_if_blank:nF { #2 }
5595        {
5596          \@@_add_to_colors_seq:en
5597            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5598            { \@@_cartesian_color:nn { - } { #3 } }
5599        }
5600    }
```

Here is an example : \@@_rectanglecolor{red!15}{2-3}{5-6}

```
5601  \NewDocumentCommand \@@_rectanglecolor { O { } m m m }
5602    {
5603      \tl_if_blank:nF { #2 }
5604        {
5605          \@@_add_to_colors_seq:en
5606            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5607            { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5608        }
5609    }
```

The last argument is the radius of the corners of the rectangle.

```
5610  \NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
5611    {
5612      \tl_if_blank:nF { #2 }
5613        {
5614          \@@_add_to_colors_seq:en
5615            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5616            { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5617        }
5618    }
```

The last argument is the radius of the corners of the rectangle.

```
5619  \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5620    {
5621      \@@_cut_on_hyphen:w #1 \q_stop
5622      \tl_clear_new:N \l_@@_tmpc_tl
5623      \tl_clear_new:N \l_@@_tmpd_tl
5624      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5625      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5626      \@@_cut_on_hyphen:w #2 \q_stop
5627      \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5628      \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.

```
5629      \@@_cartesian_path:n { #3 }
5630    }
```

Here is an example : \@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}

```
5631  \NewDocumentCommand \@@_cellcolor { O { } m m }
5632    {
5633      \clist_map_inline:nn { #3 }
5634        { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5635    }
```

```
5636  \NewDocumentCommand \@@_chessboardcolors { O { } m m  }
5637    {
5638      \int_step_inline:nn \c@iRow
5639        {
5640          \int_step_inline:nn \c@jCol
5641            {
5642              \int_if_even:nTF { ####1 + ##1 }
5643                { \@@_cellcolor [ #1 ] { #2 } }
5644                { \@@_cellcolor [ #1 ] { #3 } }
5645              { ##1 - ####1 }
5646            }
5647        }
5648    }
```

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".

```
5649  \NewDocumentCommand \@@_arraycolor { O { } m }
5650    {
5651      \@@_rectanglecolor [ #1 ] { #2 }
5652        { 1 - 1 }
5653        { \int_use:N \c@iRow - \int_use:N \c@jCol }
5654    }
```

```
5655  \keys_define:nn { nicematrix / rowcolors }
5656    {
5657      respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5658      respect-blocks .default:n = true ,
5659      cols .tl_set:N = \l_@@_cols_tl ,
5660      restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5661      restart .default:n = true ,
5662      unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5663    }
```

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 commmand `\@@_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*.

```
5664  \NewDocumentCommand \@@_rowlistcolors { O { } m m O { } }
5665    {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
5666      \group_begin:
5667      \seq_clear_new:N \l_@@_colors_seq
5668      \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5669      \tl_clear_new:N \l_@@_cols_tl
5670      \cs_set_nopar:Npn \l_@@_cols_tl { - }
5671      \keys_set:nn { nicematrix / rowcolors } { #4 }
```

The counter `\l_@@_color_int` will be the rank of the current color in the list of colors (modulo the length of the list).

```
5672      \int_zero_new:N \l_@@_color_int
5673      \int_set_eq:NN \l_@@_color_int \c_one_int
5674      \bool_if:NT \l_@@_respect_blocks_bool
5675        {
```

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`).

```
5676          \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5677          \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5678            { \@@_not_in_exterior_p:nnnnn ##1 }
5679        }
5680      \pgfpicture
5681      \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5682      \clist_map_inline:nn { #2 }
5683        {
5684          \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5685          \tl_if_in:NnTF \l_tmpa_tl { - }
5686            { \@@_cut_on_hyphen:w ##1 \q_stop }
5687            { \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.

```
5688          \int_set:Nn \l_tmpa_int \l_tmpa_tl
5689          \int_set:Nn \l_@@_color_int
5690            { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5691          \int_zero_new:N \l_@@_tmpc_int
5692          \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5693          \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5694            {
```

We will compute in `\l_tmpb_int` the last row of the "block".

```
5695              \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).

```
5696              \bool_if:NT \l_@@_respect_blocks_bool
5697                {
5698                  \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5699                    { \@@_intersect_our_row_p:nnnnn ####1 }
5700                  \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```

Now, the last row of the block is computed in `\l_tmpb_int`.

```
5701                }
5702              \tl_set:No \l_@@_rows_tl
5703                { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

`\l_@@_tmpc_tl` will be the color that we will use.

```
5704              \tl_clear_new:N \l_@@_color_tl
5705              \tl_set:Ne \l_@@_color_tl
5706                {
5707                  \@@_color_index:n
5708                    {
5709                      \int_mod:nn
5710                        { \l_@@_color_int - 1 }
5711                        { \seq_count:N \l_@@_colors_seq }
5712                      + 1
5713                    }
5714                }
5715              \tl_if_empty:NF \l_@@_color_tl
5716                {
5717                  \@@_add_to_colors_seq:ee
5718                    { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5719                    { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5720                }
5721              \int_incr:N \l_@@_color_int
5722              \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5723            }
5724        }
5725      \endpgfpicture
```

```
5726        \group_end:
5727    }
```

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.

```
5728  \cs_new:Npn \@@_color_index:n #1
5729    {
```

Be careful: this command `\@@_color_index:n` must be "*fully expandable*".

```
5730        \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5731          { \@@_color_index:n { #1 - 1 } }
5732          { \seq_item:Nn \l_@@_colors_seq { #1 } }
5733    }
```

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.

```
5734  \NewDocumentCommand \@@_rowcolors { O { } m m m }
5735    { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5736  \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5737    {
5738        \int_compare:nNnT { #3 } > \l_tmpb_int
5739          { \int_set:Nn \l_tmpb_int { #3 } }
5740    }
```

```
5741  \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5742    {
5743        \int_if_zero:nTF { #4 }
5744          \prg_return_false:
5745          {
5746            \int_compare:nNnTF { #2 } > \c@jCol
5747              \prg_return_false:
5748              \prg_return_true:
5749          }
5750    }
```

The following command return `true` when the block intersects the row `\l_tmpa_int`.

```
5751  \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5752    {
5753        \int_compare:nNnTF { #1 } > \l_tmpa_int
5754          \prg_return_false:
5755          {
5756            \int_compare:nNnTF \l_tmpa_int > { #3 }
5757              \prg_return_false:
5758              \prg_return_true:
5759          }
5760    }
```

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`).

```
5761  \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5762    {
5763        \dim_compare:nNnTF { #1 } = \c_zero_dim
```

```
5764        {
5765          \bool_if:NTF
5766            \l_@@_nocolor_used_bool
5767            \@@_cartesian_path_normal_ii:
5768            {
5769              \clist_if_empty:NTF \l_@@_corners_cells_clist
5770                { \@@_cartesian_path_normal_i:n { #1 } }
5771                \@@_cartesian_path_normal_ii:
5772            }
5773        }
5774        { \@@_cartesian_path_normal_i:n { #1 } }
5775    }
```

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.

```
5776 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5777   {
5778      \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```

We begin the loop over the columns.

```
5779      \clist_map_inline:Nn \l_@@_cols_tl
5780        {
5781          \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5782          \tl_if_in:NnTF \l_tmpa_tl { - }
5783            { \@@_cut_on_hyphen:w ##1 \q_stop }
5784            { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5785          \tl_if_empty:NTF \l_tmpa_tl
5786            { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5787            {
5788              \str_if_eq:eeT \l_tmpa_tl { * }
5789                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5790            }
5791          \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5792            { \@@_error:n { Invalid~col~number } }
5793          \tl_if_empty:NTF \l_tmpb_tl
5794            { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5795            {
5796              \str_if_eq:eeT \l_tmpb_tl { * }
5797                { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5798            }
5799          \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5800            { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```

`\l_@@_tmpc_tl` will contain the number of column.

```
5801          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5802          \@@_qpoint:n { col - \l_tmpa_tl }
5803          \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5804            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5805            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5806          \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 }  }
5807          \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5808          \clist_map_inline:Nn \l_@@_rows_tl
5809            {
5810              \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5811              \tl_if_in:NnTF \l_tmpa_tl { - }
5812                { \@@_cut_on_hyphen:w ####1 \q_stop }
5813                { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5814              \tl_if_empty:NTF \l_tmpa_tl
5815                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5816                {
5817                  \str_if_eq:eeT \l_tmpa_tl { * }
5818                    { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
```

138

```
5819                            }
5820                \tl_if_empty:NTF \l_tmpb_tl
5821                  { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5822                  {
5823                    \str_if_eq:eeT \l_tmpb_tl { * }
5824                      { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5825                  }
5826                \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5827                  { \@@_error:n { Invalid~row~number } }
5828                \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5829                  { \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.

```
5830                \cs_if_exist:cF
5831                  { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5832                  {
5833                    \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5834                    \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5835                    \@@_qpoint:n { row - \l_tmpa_tl }
5836                    \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5837                    \pgfpathrectanglecorners
5838                      { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5839                      { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5840                  }
5841              }
5842          }
5843      }
```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key corners is used).

```
5844  \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5845    {
5846      \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5847      \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5848      \clist_map_inline:Nn \l_@@_cols_tl
5849        {
5850          \@@_qpoint:n { col - ##1 }
5851          \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5852            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5853            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5854          \@@_qpoint:n { col - \int_eval:n { ##1 + 1 }  }
5855          \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5856          \clist_map_inline:Nn \l_@@_rows_tl
5857            {
5858              \@@_if_in_corner:nF { ####1 - ##1 }
5859                {
5860                  \@@_qpoint:n { row - \int_eval:n { ####1 + 1 } }
5861                  \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5862                  \@@_qpoint:n { row - ####1 }
5863                  \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5864                  \cs_if_exist:cF { @@ _ nocolor _ ####1 - ##1 }
5865                    {
5866                      \pgfpathrectanglecorners
5867                        { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5868                        { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5869                    }
5870                }
5871            }
5872        }
5873    }
```

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).

```
5874 \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

```
5875 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5876   {
5877     \bool_set_true:N \l_@@_nocolor_used_bool
5878     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5879     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```
We begin the loop over the columns.
```
5880     \clist_map_inline:Nn \l_@@_rows_tl
5881       {
5882         \clist_map_inline:Nn \l_@@_cols_tl
5883           { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5884       }
5885   }
```

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.

```
5886 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5887   {
5888     \clist_set_eq:NN \l_tmpa_clist #1
5889     \clist_clear:N #1
5890     \clist_map_inline:Nn \l_tmpa_clist
5891       {
5892         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5893         \tl_if_in:NnTF \l_tmpa_tl { - }
5894           { \@@_cut_on_hyphen:w ##1 \q_stop }
5895           { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5896         \bool_lazy_or:nnT
5897           { \str_if_eq_p:ee \l_tmpa_tl { * } }
5898           { \tl_if_blank_p:o \l_tmpa_tl }
5899           { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5900         \bool_lazy_or:nnT
5901           { \str_if_eq_p:ee \l_tmpb_tl { * } }
5902           { \tl_if_blank_p:o \l_tmpb_tl }
5903           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5904         \int_compare:nNnT \l_tmpb_tl > #2
5905           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5906         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5907           { \clist_put_right:Nn #1 { ####1 } }
5908       }
5909   }
```

The following command will be linked to \cellcolor in the tabular.

```
5910 \NewDocumentCommand \@@_cellcolor_tabular { O { } m }
5911   {
5912     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5913       {
```
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).
```
5914         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5915           { \int_use:N \c@iRow - \int_use:N \c@jCol }
5916       }
5917     \ignorespaces
5918   }
```

The following command will be linked to `\rowcolor` in the tabular.

```
5919 \NewDocumentCommand \@@_rowcolor_tabular { O { } m }
5920   {
5921     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5922       {
5923         \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5924           { \int_use:N \c@iRow - \int_use:N \c@jCol }
5925           { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5926       }
5927     \ignorespaces
5928   }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
5929 \NewDocumentCommand { \@@_rowcolors_tabular } { O { } m m }
5930   { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
5931 \NewDocumentCommand { \@@_rowlistcolors_tabular } { O { } m O { } }
5932   {
5933     \peek_remove_spaces:n
5934       { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5935   }
```

```
5936 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5937   {
```

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`.

```
5938     \seq_gclear:N \g_tmpa_seq
5939     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5940       { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5941     \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`.

```
5942     \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5943       {
5944         { \int_use:N \c@iRow }
5945         { \exp_not:n { #1 } }
5946         { \exp_not:n { #2 } }
5947         { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5948       }
5949   }
```

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`).

```
5950 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5951   {
5952     \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).

```
5953        { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } } }
5954        {
5955          \tl_gput_right:Ne \g_@@_pre_code_before_tl
5956            {
5957              \@@_rowlistcolors
5958                [ \exp_not:n { #2 } ]
5959                { #1 - \int_eval:n { \c@iRow - 1 } }
5960                { \exp_not:n { #3 } }
5961                [ \exp_not:n { #4 } ]
5962            }
5963        }
5964    }
```

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.

```
5965 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5966   {
5967     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5968       { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5969     \seq_gclear:N \g_@@_rowlistcolors_seq
5970   }
```

```
5971 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5972   {
5973     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5974       { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5975   }
```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row i until the end of the tabular.

```
5976 \NewDocumentCommand \@@_columncolor_preamble { O { } m }
5977   {
```

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).

```
5978     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5979       {
```

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).

```
5980         \tl_gput_left:Ne \g_@@_pre_code_before_tl
5981           {
5982             \exp_not:N \columncolor [ #1 ]
5983               { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5984           }
5985       }
5986   }
```

```
5987 \hook_gput_code:nnn { begindocument } { . }
5988   {
5989     \IfPackageLoadedTF { colortbl }
5990       {
5991         \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5992         \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5993         \cs_new_protected:Npn \@@_revert_colortbl:
```

```
5994              {
5995                \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5996                  {
5997                    \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5998                    \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5999                  }
6000              }
6001            }
6002          { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6003      }


6004  \cs_new_protected:Npn \@@_EmptyColumn:n #1
6005    {
6006      \clist_map_inline:nn { #1 }
6007        {
6008          \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6009            { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6010          \columncolor { nocolor } { ##1 }
6011        }
6012    }
6013  \cs_new_protected:Npn \@@_EmptyRow:n #1
6014    {
6015      \clist_map_inline:nn { #1 }
6016        {
6017          \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6018            { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6019          \rowcolor { nocolor } { ##1 }
6020        }
6021    }
```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumntype` 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`.

```
6022  \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`.

```
6023  \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6024    {
6025      \int_if_zero:nTF \l_@@_first_col_int
6026        { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6027        {
6028          \int_if_zero:nTF \c@jCol
6029            {
6030              \int_compare:nNnF \c@iRow = { -1 }
6031                { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6032            }
6033            { \@@_OnlyMainNiceMatrix_i:n { #1 } }
```

```
6034        }
6035    }
```

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.

```
6036 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6037    {
6038      \int_if_zero:nF \c@iRow
6039        {
6040          \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6041            {
6042              \int_compare:nNnT \c@jCol > \c_zero_int
6043                { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6044            }
6045        }
6046    }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```
6047 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6048    {
6049      \IfPackageLoadedTF { tikz }
6050        {
6051          \IfPackageLoadedTF { booktabs }
6052            { #2 }
6053            { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6054        }
6055        { \@@_error:nn { TopRule~without~tikz } { #1 } }
6056    }
6057 \NewExpandableDocumentCommand { \@@_TopRule } {  }
6058    { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }

6059 \cs_new:Npn \@@_TopRule_i:
6060    {
6061      \noalign \bgroup
6062        \peek_meaning:NTF [
6063          { \@@_TopRule_ii: }
6064          { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6065    }
6066 \NewDocumentCommand \@@_TopRule_ii: { o }
6067    {
6068      \tl_gput_right:Ne \g_@@_pre_code_after_tl
6069        {
6070          \@@_hline:n
6071            {
6072              position = \int_eval:n { \c@iRow + 1 } ,
6073              tikz =
6074                {
6075                  line~width = #1 ,
6076                  yshift =  0.25 \arrayrulewidth ,
6077                  shorten~< = - 0.5 \arrayrulewidth
6078                } ,
6079              total-width = #1
6080            }
6081        }
6082      \skip_vertical:n { \belowrulesep + #1 }
6083      \egroup
6084    }
```

```
6085  \NewExpandableDocumentCommand { \@@_BottomRule } { }
6086    { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }
6087  \cs_new:Npn \@@_BottomRule_i:
6088    {
6089      \noalign \bgroup
6090        \peek_meaning:NTF [
6091          { \@@_BottomRule_ii: }
6092          { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6093    }
6094  \NewDocumentCommand \@@_BottomRule_ii: { o }
6095    {
6096      \tl_gput_right:Ne \g_@@_pre_code_after_tl
6097        {
6098          \@@_hline:n
6099            {
6100              position = \int_eval:n { \c@iRow + 1 } ,
6101              tikz =
6102                {
6103                  line~width = #1 ,
6104                  yshift =  0.25 \arrayrulewidth ,
6105                  shorten~< = - 0.5 \arrayrulewidth
6106                } ,
6107              total-width = #1 ,
6108            }
6109        }
6110      \skip_vertical:N \aboverulesep
6111      \@@_create_row_node_i:
6112      \skip_vertical:n { #1 }
6113      \egroup
6114    }
6115  \NewExpandableDocumentCommand { \@@_MidRule } { }
6116    { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }
6117  \cs_new:Npn \@@_MidRule_i:
6118    {
6119      \noalign \bgroup
6120        \peek_meaning:NTF [
6121          { \@@_MidRule_ii: }
6122          { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6123    }
6124  \NewDocumentCommand \@@_MidRule_ii: { o }
6125    {
6126      \skip_vertical:N \aboverulesep
6127      \@@_create_row_node_i:
6128      \tl_gput_right:Ne \g_@@_pre_code_after_tl
6129        {
6130          \@@_hline:n
6131            {
6132              position = \int_eval:n { \c@iRow + 1 } ,
6133              tikz =
6134                {
6135                  line~width = #1 ,
6136                  yshift =  0.25 \arrayrulewidth ,
6137                  shorten~< = - 0.5 \arrayrulewidth
6138                } ,
6139              total-width = #1 ,
6140            }
6141        }
6142      \skip_vertical:n { \belowrulesep + #1 }
6143      \egroup
6144    }
```

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.

```
6145 \keys_define:nn { nicematrix / Rules }
6146   {
6147     position .int_set:N = \l_@@_position_int ,
6148     position .value_required:n = true ,
6149     start .int_set:N = \l_@@_start_int ,
6150     end .code:n =
6151       \bool_lazy_or:nnTF
6152         { \tl_if_empty_p:n { #1 } }
6153         { \str_if_eq_p:ee { #1 } { last } }
6154         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6155         { \int_set:Nn \l_@@_end_int { #1 } } }
6156   }
```

It's possible that the rule won't be drawn continuously from **start** ot **end** because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```
6157 \keys_define:nn { nicematrix / RulesBis }
6158   {
6159     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6160     multiplicity .initial:n = 1 ,
6161     dotted .bool_set:N = \l_@@_dotted_bool ,
6162     dotted .initial:n = false ,
6163     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]`.

```
6164     color .code:n =
6165       \@@_set_CT@arc@:n { #1 }
6166       \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6167     color .value_required:n = true ,
6168     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6169     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.

```
6170     tikz .code:n =
6171       \IfPackageLoadedTF { tikz }
6172         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6173         { \@@_error:n { tikz~without~tikz } } ,
6174     tikz .value_required:n = true ,
6175     total-width .dim_set:N = \l_@@_rule_width_dim ,
6176     total-width .value_required:n = true ,
6177     width .meta:n = { total-width = #1 } ,
6178     unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6179   }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```
6180 \cs_new_protected:Npn \@@_vline:n #1
6181   {
```

The group is for the options.

```
6182        \group_begin:
6183        \int_set_eq:NN \l_@@_end_int \c@iRow
6184        \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).

```
6185        \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6186          \@@_vline_i:
6187        \group_end:
6188      }
6189  \cs_new_protected:Npn \@@_vline_i:
6190      {
```

`\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`.

```
6191        \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6192        \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6193          \l_tmpa_tl
6194          {
```

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.

```
6195          \bool_gset_true:N \g_tmpa_bool
6196          \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6197            { \@@_test_vline_in_block:nnnnn ##1 }
6198          \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6199            { \@@_test_vline_in_block:nnnnn ##1 }
6200          \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6201            { \@@_test_vline_in_stroken_block:nnnn ##1 }
6202          \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6203          \bool_if:NTF \g_tmpa_bool
6204            {
6205              \int_if_zero:nT \l_@@_local_start_int
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
6206                { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6207            }
6208            {
6209              \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6210                {
6211                  \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6212                  \@@_vline_ii:
6213                  \int_zero:N \l_@@_local_start_int
6214                }
6215            }
6216        }
6217      \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6218        {
6219          \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6220          \@@_vline_ii:
6221        }
6222    }


6223  \cs_new_protected:Npn \@@_test_in_corner_v:
6224      {
6225        \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6226          {
6227            \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
```

```
6228          { \bool_set_false:N \g_tmpa_bool }
6229        }
6230        {
6231          \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6232            {
6233              \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6234                { \bool_set_false:N \g_tmpa_bool }
6235                {
6236                  \@@_if_in_corner:nT
6237                    { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6238                    { \bool_set_false:N \g_tmpa_bool }
6239                }
6240            }
6241        }
6242    }


6243 \cs_new_protected:Npn \@@_vline_ii:
6244   {
6245     \tl_clear:N \l_@@_tikz_rule_tl
6246     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6247     \bool_if:NTF \l_@@_dotted_bool
6248       \@@_vline_iv:
6249       {
6250         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6251           \@@_vline_iii:
6252           \@@_vline_v:
6253       }
6254   }
```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```
6255 \cs_new_protected:Npn \@@_vline_iii:
6256   {
6257     \pgfpicture
6258     \pgfrememberpicturepositiononpagetrue
6259     \pgf@relevantforpicturesizefalse
6260     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6261     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6262     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6263     \dim_set:Nn \l_tmpb_dim
6264       {
6265         \pgf@x
6266         - 0.5 \l_@@_rule_width_dim
6267         +
6268         ( \arrayrulewidth * \l_@@_multiplicity_int
6269           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6270       }
6271     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6272     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6273     \bool_lazy_all:nT
6274       {
6275         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6276         { \cs_if_exist_p:N \CT@drsc@ }
6277         { ! \tl_if_blank_p:o \CT@drsc@ }
6278       }
6279       {
6280         \group_begin:
6281         \CT@drsc@
6282         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6283         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6284         \dim_set:Nn \l_@@_tmpd_dim
6285           {
6286             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
```

```
6287              * ( \l_@@_multiplicity_int - 1 )
6288            }
6289          \pgfpathrectanglecorners
6290            { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6291            { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6292          \pgfusepath { fill }
6293          \group_end:
6294        }
6295      \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6296      \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6297      \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6298        {
6299          \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6300          \dim_sub:Nn \l_tmpb_dim \doublerulesep
6301          \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6302          \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6303        }
6304      \CT@arc@
6305      \pgfsetlinewidth { 1.1 \arrayrulewidth }
6306      \pgfsetrectcap
6307      \pgfusepathqstroke
6308      \endpgfpicture
6309    }
```

The following code is for the case of a dotted rule (with our system of rounded dots).

```
6310 \cs_new_protected:Npn \@@_vline_iv:
6311   {
6312     \pgfpicture
6313     \pgfrememberpicturepositiononpagetrue
6314     \pgf@relevantforpicturesizefalse
6315     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6316     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6317     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6318     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6319     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6320     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6321     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6322     \CT@arc@
6323     \@@_draw_line:
6324     \endpgfpicture
6325   }
```

The following code is for the case when the user uses the key tikz.

```
6326 \cs_new_protected:Npn \@@_vline_v:
6327   {
6328     \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.

```
6329     \CT@arc@
6330     \tl_if_empty:NF \l_@@_rule_color_tl
6331       { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6332     \pgfrememberpicturepositiononpagetrue
6333     \pgf@relevantforpicturesizefalse
6334     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6335     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6336     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6337     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6338     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6339     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6340     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6341     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
```

149

```
6342        ( \l_tmpb_dim , \l_tmpa_dim ) --
6343        ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6344      \end { tikzpicture }
6345    }
```

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).

```
6346 \cs_new_protected:Npn \@@_draw_vlines:
6347    {
6348      \int_step_inline:nnn
6349        { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6350        {
6351          \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6352            \c@jCol
6353            { \int_eval:n { \c@jCol + 1 } }
6354        }
6355        {
6356          \str_if_eq:eeF \l_@@_vlines_clist { all }
6357            { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6358            { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6359        }
6360    }
```

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}`.

```
6361 \cs_new_protected:Npn \@@_hline:n #1
6362    {
```

The group is for the options.

```
6363      \group_begin:
6364      \int_zero_new:N \l_@@_end_int
6365      \int_set_eq:NN \l_@@_end_int \c@jCol
6366      \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6367      \@@_hline_i:
6368      \group_end:
6369    }
```

```
6370 \cs_new_protected:Npn \@@_hline_i:
6371    {
6372      \int_zero_new:N \l_@@_local_start_int
6373      \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`.

```
6374      \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6375      \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6376        \l_tmpb_tl
6377        {
```

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.

```
6378          \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6379          \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6380            { \@@_test_hline_in_block:nnnnn ##1 }
```

```
6381        \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6382          { \@@_test_hline_in_block:nnnnn ##1 }
6383        \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6384          { \@@_test_hline_in_stroken_block:nnnn ##1 }
6385        \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6386        \bool_if:NTF \g_tmpa_bool
6387          {
6388            \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.

```
6389              { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6390          }
6391          {
6392            \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6393              {
6394                \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6395                \@@_hline_ii:
6396                \int_zero:N \l_@@_local_start_int
6397              }
6398          }
6399      }
6400    \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6401      {
6402        \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6403        \@@_hline_ii:
6404      }
6405  }


6406 \cs_new_protected:Npn \@@_test_in_corner_h:
6407   {
6408     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6409       {
6410         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6411           { \bool_set_false:N \g_tmpa_bool }
6412       }
6413       {
6414         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6415           {
6416             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6417               { \bool_set_false:N \g_tmpa_bool }
6418               {
6419                 \@@_if_in_corner:nT
6420                   { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6421                   { \bool_set_false:N \g_tmpa_bool }
6422               }
6423           }
6424       }
6425   }


6426 \cs_new_protected:Npn \@@_hline_ii:
6427   {
6428     \tl_clear:N \l_@@_tikz_rule_tl
6429     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6430     \bool_if:NTF \l_@@_dotted_bool
6431       \@@_hline_iv:
6432       {
6433         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6434           \@@_hline_iii:
6435           \@@_hline_v:
6436       }
6437   }
```

First the case of a standard rule (without the keys `dotted` and `tikz`).

```
6438 \cs_new_protected:Npn \@@_hline_iii:
6439   {
6440     \pgfpicture
6441     \pgfrememberpicturepositiononpagetrue
6442     \pgf@relevantforpicturesizefalse
6443     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6444     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6445     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6446     \dim_set:Nn \l_tmpb_dim
6447       {
6448         \pgf@y
6449         - 0.5 \l_@@_rule_width_dim
6450         +
6451         ( \arrayrulewidth * \l_@@_multiplicity_int
6452           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6453       }
6454     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6455     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6456     \bool_lazy_all:nT
6457       {
6458         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6459         { \cs_if_exist_p:N \CT@drsc@ }
6460         { ! \tl_if_blank_p:o \CT@drsc@ }
6461       }
6462       {
6463         \group_begin:
6464         \CT@drsc@
6465         \dim_set:Nn \l_@@_tmpd_dim
6466           {
6467             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6468             * ( \l_@@_multiplicity_int - 1 )
6469           }
6470         \pgfpathrectanglecorners
6471           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6472           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6473         \pgfusepathqfill
6474         \group_end:
6475       }
6476     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6477     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6478     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6479       {
6480         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6481         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6482         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6483         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6484       }
6485     \CT@arc@
6486     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6487     \pgfsetrectcap
6488     \pgfusepathqstroke
6489     \endpgfpicture
6490   }
```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{bmatrix}$$

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6491  \cs_new_protected:Npn \@@_hline_iv:
6492    {
6493      \pgfpicture
6494      \pgfrememberpicturepositiononpagetrue
6495      \pgf@relevantforpicturesizefalse
6496      \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6497      \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6498      \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6499      \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6500      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6501      \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6502        {
6503          \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6504          \bool_if:NF \g_@@_delims_bool
6505            { \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.

```
6506          \tl_if_eq:NnF \g_@@_left_delim_tl (
6507            { \dim_add:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_xdots_inter_dim } }
6508        }
6509      \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6510      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6511      \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6512        {
6513          \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6514          \bool_if:NF \g_@@_delims_bool
6515            { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6516          \tl_if_eq:NnF \g_@@_right_delim_tl )
6517            { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6518        }
6519      \CT@arc@
6520      \@@_draw_line:
6521      \endpgfpicture
6522    }
```

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`).

```
6523  \cs_new_protected:Npn \@@_hline_v:
6524    {
6525      \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.

```
6526      \CT@arc@
6527      \tl_if_empty:NF \l_@@_rule_color_tl
```

```
6528          { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6529      \pgfrememberpicturepositiononpagetrue
6530      \pgf@relevantforpicturesizefalse
6531      \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6532      \dim_set_eq:NN \l_tmpa_dim \pgf@x
6533      \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6534      \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6535      \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6536      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6537      \exp_args:No \tikzset \l_@@_tikz_rule_tl
6538      \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6539        ( \l_tmpa_dim , \l_tmpb_dim ) --
6540        ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6541      \end { tikzpicture }
6542    }
```

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).

```
6543 \cs_new_protected:Npn \@@_draw_hlines:
6544   {
6545     \int_step_inline:nnn
6546       { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6547       {
6548         \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6549           \c@iRow
6550           { \int_eval:n { \c@iRow + 1 } }
6551       }
6552       {
6553         \str_if_eq:eeF \l_@@_hlines_clist { all }
6554           { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6555           { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6556       }
6557   }
```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of nicematrix.

```
6558 \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.

```
6559 \cs_set:Npn \@@_Hline_i:n #1
6560   {
6561     \peek_remove_spaces:n
6562       {
6563         \peek_meaning:NTF \Hline
6564           { \@@_Hline_ii:nn { #1 + 1 } }
6565           { \@@_Hline_iii:n { #1 } } }
6566       }
6567   }
6568 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6569 \cs_set:Npn \@@_Hline_iii:n #1
6570   { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6571 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6572   {
6573     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6574     \skip_vertical:N \l_@@_rule_width_dim
6575     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6576       {
6577         \@@_hline:n
6578           {
6579             multiplicity = #1 ,
6580             position = \int_eval:n { \c@iRow + 1 } ,
```

```
6581          total-width = \dim_use:N \l_@@_rule_width_dim ,
6582          #2
6583        }
6584    }
6585  \egroup
6586  }
```

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`).

```
6587 \cs_new_protected:Npn \@@_custom_line:n #1
6588   {
6589     \str_clear_new:N \l_@@_command_str
6590     \str_clear_new:N \l_@@_ccommand_str
6591     \str_clear_new:N \l_@@_letter_str
6592     \tl_clear_new:N \l_@@_other_keys_tl
6593     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```
6594     \bool_lazy_all:nTF
6595       {
6596         { \str_if_empty_p:N \l_@@_letter_str }
6597         { \str_if_empty_p:N \l_@@_command_str }
6598         { \str_if_empty_p:N \l_@@_ccommand_str }
6599       }
6600       { \@@_error:n { No~letter~and~no~command } }
6601       { \@@_custom_line_i:o \l_@@_other_keys_tl }
6602   }
6603 \keys_define:nn { nicematrix / custom-line }
6604   {
6605     letter .str_set:N = \l_@@_letter_str ,
6606     letter .value_required:n = true ,
6607     command .str_set:N = \l_@@_command_str ,
6608     command .value_required:n = true ,
6609     ccommand .str_set:N = \l_@@_ccommand_str ,
6610     ccommand .value_required:n = true ,
6611   }


6612 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6613 \cs_new_protected:Npn \@@_custom_line_i:n #1
6614   {
```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```
6615     \bool_set_false:N \l_@@_tikz_rule_bool
6616     \bool_set_false:N \l_@@_dotted_rule_bool
6617     \bool_set_false:N \l_@@_color_bool

6618     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6619     \bool_if:NT \l_@@_tikz_rule_bool
6620       {
6621         \IfPackageLoadedF { tikz }
6622           { \@@_error:n { tikz~in~custom-line~without~tikz } }
6623         \bool_if:NT \l_@@_color_bool
6624           { \@@_error:n { color~in~custom-line~with~tikz } }
6625       }
```

```
6626      \bool_if:NT \l_@@_dotted_rule_bool
6627        {
6628          \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6629            { \@@_error:n { key~multiplicity~with~dotted } }
6630        }
6631      \str_if_empty:NF \l_@@_letter_str
6632        {
6633          \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6634            { \@@_error:n { Several~letters } }
6635            {
6636              \tl_if_in:NoTF
6637                \c_@@_forbidden_letters_str
6638                \l_@@_letter_str
6639                { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6640                {
```

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.

```
6641                  \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6642                    { \@@_v_custom_line:n { #1 } }
6643                }
6644            }
6645        }
6646      \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6647      \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6648    }
6649  \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6650  \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.

```
6651  \keys_define:nn { nicematrix / custom-line-bis }
6652    {
6653      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6654      multiplicity .initial:n = 1 ,
6655      multiplicity .value_required:n = true ,
6656      color .code:n = \bool_set_true:N \l_@@_color_bool ,
6657      color .value_required:n = true ,
6658      tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6659      tikz .value_required:n = true ,
6660      dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6661      dotted .value_forbidden:n = true ,
6662      total-width .code:n = { } ,
6663      total-width .value_required:n = true ,
6664      width .code:n = { } ,
6665      width .value_required:n = true ,
6666      sep-color .code:n = { } ,
6667      sep-color .value_required:n = true ,
6668      unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6669    }
```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
6670  \bool_new:N \l_@@_dotted_rule_bool
6671  \bool_new:N \l_@@_tikz_rule_bool
6672  \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`.

```
6673  \keys_define:nn { nicematrix / custom-line-width }
6674    {
6675      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6676      multiplicity .initial:n = 1 ,
6677      multiplicity .value_required:n = true ,
6678      tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6679      total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6680                            \bool_set_true:N \l_@@_total_width_bool ,
6681      total-width .value_required:n = true ,
6682      width .meta:n = { total-width = #1 } ,
6683      dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6684    }
```

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`).

```
6685  \cs_new_protected:Npn \@@_h_custom_line:n #1
6686    {
```

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`).

```
6687      \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6688      \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6689    }
```

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`).

```
6690  \cs_new_protected:Npn \@@_c_custom_line:n #1
6691    {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6692      \exp_args:Nc \NewExpandableDocumentCommand
6693        { nicematrix - \l_@@_ccommand_str }
6694        { O { } m }
6695        {
6696          \noalign
6697            {
6698              \@@_compute_rule_width:n { #1 , ##1 }
6699              \skip_vertical:n { \l_@@_rule_width_dim }
6700              \clist_map_inline:nn
6701                { ##2 }
6702                { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6703            }
6704        }
6705      \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6706    }
```

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*.

```
6707  \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6708    {
6709      \tl_if_in:nnTF { #2 } { - }
6710        { \@@_cut_on_hyphen:w #2 \q_stop }
6711        { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6712      \tl_gput_right:Ne \g_@@_pre_code_after_tl
6713        {
6714          \@@_hline:n
6715            {
6716              #1 ,
6717              start = \l_tmpa_tl ,
```

```
6718                end = \l_tmpb_tl ,
6719                position = \int_eval:n { \c@iRow + 1 } ,
6720                total-width = \dim_use:N \l_@@_rule_width_dim
6721              }
6722          }
6723      }
6724  \cs_new_protected:Npn \@@_compute_rule_width:n #1
6725    {
6726      \bool_set_false:N \l_@@_tikz_rule_bool
6727      \bool_set_false:N \l_@@_total_width_bool
6728      \bool_set_false:N \l_@@_dotted_rule_bool
6729      \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6730      \bool_if:NF \l_@@_total_width_bool
6731        {
6732          \bool_if:NTF \l_@@_dotted_rule_bool
6733            { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6734            {
6735              \bool_if:NF \l_@@_tikz_rule_bool
6736                {
6737                  \dim_set:Nn \l_@@_rule_width_dim
6738                    {
6739                      \arrayrulewidth * \l_@@_multiplicity_int
6740                      + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6741                    }
6742                }
6743            }
6744        }
6745    }
6746  \cs_new_protected:Npn \@@_v_custom_line:n #1
6747    {
6748      \@@_compute_rule_width:n { #1 }
```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```
6749      \tl_gput_right:Ne \g_@@_array_preamble_tl
6750        { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6751      \tl_gput_right:Ne \g_@@_pre_code_after_tl
6752        {
6753          \@@_vline:n
6754            {
6755              #1 ,
6756              position = \int_eval:n { \c@jCol + 1 } ,
6757              total-width = \dim_use:N \l_@@_rule_width_dim
6758            }
6759        }
6760      \@@_rec_preamble:n
6761    }
6762  \@@_custom_line:n
6763    { 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.

```
6764  \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6765    {
6766      \int_compare:nNnT \l_tmpa_tl > { #1 }
6767        {
6768          \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6769            {
6770              \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
```

```
6771                {
6772                  \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6773                    { \bool_gset_false:N \g_tmpa_bool }
6774                }
6775            }
6776        }
6777    }
```

The same for vertical rules.

```
6778  \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6779    {
6780      \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6781        {
6782          \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6783            {
6784              \int_compare:nNnT \l_tmpb_tl > { #2 }
6785                {
6786                  \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6787                    { \bool_gset_false:N \g_tmpa_bool }
6788                }
6789            }
6790        }
6791    }
6792  \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6793    {
6794      \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6795        {
6796          \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6797            {
6798              \int_compare:nNnTF \l_tmpa_tl = { #1 }
6799                { \bool_gset_false:N \g_tmpa_bool }
6800                {
6801                  \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6802                    { \bool_gset_false:N \g_tmpa_bool }
6803                }
6804            }
6805        }
6806    }
6807  \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6808    {
6809      \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6810        {
6811          \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6812            {
6813              \int_compare:nNnTF \l_tmpb_tl = { #2 }
6814                { \bool_gset_false:N \g_tmpa_bool }
6815                {
6816                  \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6817                    { \bool_gset_false:N \g_tmpa_bool }
6818                }
6819            }
6820        }
6821    }
```

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.

```
6822 \cs_new_protected:Npn \@@_compute_corners:
6823   {
6824     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6825       { \@@_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`).

```
6826     \clist_clear:N \l_@@_corners_cells_clist
6827     \clist_map_inline:Nn \l_@@_corners_clist
6828       {
6829         \str_case:nnF { ##1 }
6830           {
6831             { NW }
6832             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6833             { NE }
6834             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6835             { SW }
6836             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6837             { SE }
6838             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6839           }
6840         { \@@_error:nn { bad~corner } { ##1 } } }
6841       }
```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```
6842     \clist_if_empty:NF \l_@@_corners_cells_clist
6843       {
```

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.

```
6844         \tl_gput_right:Ne \g_@@_aux_tl
6845           {
6846             \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6847               { \l_@@_corners_cells_clist }
6848           }
6849       }
6850   }
```


```
6851 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6852   {
6853     \int_step_inline:nnn { #1 } { #3 }
6854       {
6855         \int_step_inline:nnn { #2 } { #4 }
6856           { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6857       }
6858   }
```


```
6859 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6860   {
6861     \cs_if_exist:cTF
6862       { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6863       \prg_return_true:
6864       \prg_return_false:
6865   }
```

"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.

```
6866 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6867   {
```

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.

```
6868     \bool_set_false:N \l_tmpa_bool
6869     \int_zero_new:N \l_@@_last_empty_row_int
6870     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6871     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6872       {
6873         \bool_lazy_or:nnTF
6874           {
6875             \cs_if_exist_p:c
6876               { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6877           }
6878           { \@@_if_in_block_p:nn { ##1 } { #2 } }
6879           { \bool_set_true:N \l_tmpa_bool }
6880           {
6881             \bool_if:NF \l_tmpa_bool
6882               { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6883           }
6884       }
```

Now, you determine the last empty cell in the row of number 1.

```
6885     \bool_set_false:N \l_tmpa_bool
6886     \int_zero_new:N \l_@@_last_empty_column_int
6887     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6888     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6889       {
6890         \bool_lazy_or:nnTF
6891           {
6892             \cs_if_exist_p:c
6893               { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6894           }
6895           { \@@_if_in_block_p:nn { #1 } { ##1 } }
6896           { \bool_set_true:N \l_tmpa_bool }
6897           {
6898             \bool_if:NF \l_tmpa_bool
6899               { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6900           }
6901       }
```

Now, we loop over the rows.

```
6902     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6903       {
```

We treat the row number ##1 with another loop.

```
6904         \bool_set_false:N \l_tmpa_bool
6905         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6906           {
6907             \bool_lazy_or:nnTF
6908               { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 } }
6909               { \@@_if_in_block_p:nn  { ##1 } { ####1 } }
6910               { \bool_set_true:N \l_tmpa_bool }
6911               {
6912                 \bool_if:NF \l_tmpa_bool
```

161

```
6913                    {
6914                      \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6915                      \clist_put_right:Nn
6916                        \l_@@_corners_cells_clist
6917                        { ##1 - ####1 }
6918                      \cs_set_nopar:cpn { @@ _ corner _ ##1 - ####1 } { }
6919                    }
6920                }
6921            }
6922        }
6923    }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```
6924 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6925 \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.

```
6926 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6927 \keys_define:nn { nicematrix / NiceMatrixBlock }
6928    {
6929      auto-columns-width .code:n =
6930        {
6931          \bool_set_true:N \l_@@_block_auto_columns_width_bool
6932          \dim_gzero_new:N \g_@@_max_cell_width_dim
6933          \bool_set_true:N \l_@@_auto_columns_width_bool
6934        }
6935    }
```

```
6936 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6937    {
6938      \int_gincr:N \g_@@_NiceMatrixBlock_int
6939      \dim_zero:N \l_@@_columns_width_dim
6940      \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6941      \bool_if:NT \l_@@_block_auto_columns_width_bool
6942        {
6943          \cs_if_exist:cT
6944            { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6945            {
6946              \dim_set:Nn \l_@@_columns_width_dim
6947                {
6948                  \use:c
6949                    { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6950                }
6951            }
6952        }
6953    }
```

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).

```
6954    {
6955      \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.

```
6956        { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6957        {
6958          \bool_if:NT \l_@@_block_auto_columns_width_bool
6959            {
6960              \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6961              \iow_shipout:Ne \@mainaux
6962                {
6963                  \cs_gset:cpn
6964                    { @@ _ 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.

```
6965                    { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6966                }
6967              \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6968            }
6969        }
6970      \ignorespacesafterend
6971    }
```

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).

```
6972 \cs_new_protected:Npn \@@_create_extra_nodes:
6973   {
6974     \bool_if:nTF \l_@@_medium_nodes_bool
6975       {
6976         \bool_if:NTF \l_@@_no_cell_nodes_bool
6977           { \@@_error:n { extra-nodes~with~no-cell-nodes } }
6978           {
6979             \bool_if:NTF \l_@@_large_nodes_bool
6980               \@@_create_medium_and_large_nodes:
6981               \@@_create_medium_nodes:
6982           }
6983       }
6984       {
6985         \bool_if:NT \l_@@_large_nodes_bool
6986           {
6987             \bool_if:NTF \l_@@_no_cell_nodes_bool
6988               { \@@_error:n { extra-nodes~with~no-cell-nodes } }
6989               \@@_create_large_nodes:
6990           }
6991       }
6992   }
```

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`.

```
6993 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6994   {
6995     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6996       {
6997         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6998         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6999         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
7000         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
7001       }
7002     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7003       {
7004         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
7005         \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
7006         \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
7007         \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
7008       }
```

We begin the two nested loops over the rows and the columns of the array.

```
7009     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7010       {
7011         \int_step_variable:nnNn
7012           \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.

```
7013           {
7014             \cs_if_exist:cT
7015               { 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`.

```
7016               {
7017                 \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
7018                 \dim_set:cn { l_@@_row_\@@_i: _min_dim}
7019                   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7020                 \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7021                   {
7022                     \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
7023                       { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7024                   }
```

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`.

```
7025                 \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
7026                 \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7027                   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7028                 \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7029                   {
7030                     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
```

```
7031                          { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7032                     }
7033                 }
7034             }
7035         }
```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```
7036         \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7037             {
7038                 \dim_compare:nNnT
7039                     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7040                     {
7041                         \@@_qpoint:n { row - \@@_i: - base }
7042                         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7043                         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7044                     }
7045             }
7046         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7047             {
7048                 \dim_compare:nNnT
7049                     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7050                     {
7051                         \@@_qpoint:n { col - \@@_j: }
7052                         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7053                         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7054                     }
7055             }
7056     }
```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the "medium nodes" are created.

```
7057 \cs_new_protected:Npn \@@_create_medium_nodes:
7058     {
7059         \pgfpicture
7060             \pgfrememberpicturepositiononpagetrue
7061             \pgf@relevantforpicturesizefalse
7062             \@@_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".

```
7063             \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7064             \@@_create_nodes:
7065         \endpgfpicture
7066     }
```

The command `\@@_create_large_nodes:` must be used when we want to create only the "large nodes" and not the medium ones[14]. 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:`.

```
7067 \cs_new_protected:Npn \@@_create_large_nodes:
7068     {
7069         \pgfpicture
7070             \pgfrememberpicturepositiononpagetrue
7071             \pgf@relevantforpicturesizefalse
7072             \@@_computations_for_medium_nodes:
7073             \@@_computations_for_large_nodes:
7074             \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7075             \@@_create_nodes:
```

---

[14]If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```
7076        \endpgfpicture
7077    }
7078  \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7079    {
7080      \pgfpicture
7081        \pgfrememberpicturepositiononpagetrue
7082        \pgf@relevantforpicturesizefalse
7083        \@@_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".

```
7084        \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
7085        \@@_create_nodes:
7086        \@@_computations_for_large_nodes:
7087        \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7088        \@@_create_nodes:
7089      \endpgfpicture
7090    }
```

For "large nodes", the exterior rows and columns don't interfer. 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.

```
7091  \cs_new_protected:Npn \@@_computations_for_large_nodes:
7092    {
7093      \int_set_eq:NN \l_@@_first_row_int \c_one_int
7094      \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`.

```
7095      \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7096        {
7097          \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7098            {
7099              (
7100                \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7101                \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 }  _ max _ dim }
7102              )
7103              / 2
7104            }
7105          \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7106            { l_@@_row_\@@_i: _min_dim }
7107        }
7108      \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7109        {
7110          \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7111            {
7112              (
7113                \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7114                \dim_use:c
7115                  { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7116              )
7117              / 2
7118            }
7119          \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7120            { l_@@_column _ \@@_j: _ max _ dim }
7121        }
```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```
7122      \dim_sub:cn
7123        { l_@@_column _ 1 _ min _ dim }
7124        \l_@@_left_margin_dim
7125      \dim_add:cn
7126        { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7127        \l_@@_right_margin_dim
7128    }
```

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`).

```
7129 \cs_new_protected:Npn \@@_create_nodes:
7130   {
7131     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7132       {
7133         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7134           {
```

We draw the rectangular node for the cell (`\@@_i:`-`\@@_j:`).

```
7135             \@@_pgf_rect_node:nnnnn
7136               { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7137               { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7138               { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7139               { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7140               { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7141             \str_if_empty:NF \l_@@_name_str
7142               {
7143                 \pgfnodealias
7144                   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7145                   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7146               }
7147           }
7148       }
```

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 correspondant values of n.

```
7149         \seq_map_pairwise_function:NNN
7150         \g_@@_multicolumn_cells_seq
7151         \g_@@_multicolumn_sizes_seq
7152         \@@_node_for_multicolumn:nn
7153   }
```

```
7154 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7155   {
7156     \cs_set_nopar:Npn \@@_i: { #1 }
7157     \cs_set_nopar:Npn \@@_j: { #2 }
7158   }
```

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").

```
7159 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7160   {
7161     \@@_extract_coords_values: #1 \q_stop
7162     \@@_pgf_rect_node:nnnnn
7163       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7164       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7165       { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7166       { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7167       { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7168     \str_if_empty:NF \l_@@_name_str
7169       {
7170         \pgfnodealias
7171           { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7172           { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
7173       }
7174   }
```

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).

```
7175 \keys_define:nn { nicematrix / Block / FirstPass }
7176   {
7177     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7178               \bool_set_true:N \l_@@_p_block_bool ,
7179     j .value_forbidden:n = true ,
7180     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7181     l .value_forbidden:n = true ,
7182     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7183     r .value_forbidden:n = true ,
7184     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7185     c .value_forbidden:n = true ,
7186     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7187     L .value_forbidden:n = true ,
7188     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7189     R .value_forbidden:n = true ,
7190     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7191     C .value_forbidden:n = true ,
7192     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7193     t .value_forbidden:n = true ,
7194     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7195     T .value_forbidden:n = true ,
7196     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7197     b .value_forbidden:n = true ,
7198     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7199     B .value_forbidden:n = true ,
7200     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7201     m .value_forbidden:n = true ,
7202     v-center .meta:n = m ,
7203     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7204     p .value_forbidden:n = true ,
7205     color .code:n =
7206       \@@_color:n { #1 }
7207       \tl_set_rescan:Nnn
7208         \l_@@_draw_tl
7209         { \char_set_catcode_other:N ! }
7210         { #1 } ,
7211     color .value_required:n = true ,
7212     respect-arraystretch .code:n =
7213       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7214     respect-arraystretch .value_forbidden:n = true ,
7215   }
```

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.

```
7216 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```
7217 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7218   {
```

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).

```
7219     \peek_remove_spaces:n
```

```
7220        {
7221          \tl_if_blank:nTF { #2 }
7222            { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7223            {
7224              \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7225              \@@_Block_i_czech \@@_Block_i
7226              #2 \q_stop
7227            }
7228          { #1 } { #3 } { #4 }
7229        }
7230    }
```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7231 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
```

With babel with the key czech, the character - (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command \@@_Block: to do the job because the command \@@_Block: is defined with the command \NewExpandableDocumentCommand.

```
7232 {
7233    \char_set_catcode_active:N -
7234    \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7235 }
```

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.

```
7236 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7237    {
```

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).

```
7238      \bool_lazy_or:nnTF
7239        { \tl_if_blank_p:n { #1 } }
7240        { \str_if_eq_p:ee { * } { #1 } }
7241        { \int_set:Nn \l_tmpa_int { 100 } }
7242        { \int_set:Nn \l_tmpa_int { #1 } }
7243      \bool_lazy_or:nnTF
7244        { \tl_if_blank_p:n { #2 } }
7245        { \str_if_eq_p:ee { * } { #2 } }
7246        { \int_set:Nn \l_tmpb_int { 100 } }
7247        { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7248      \int_compare:nNnTF \l_tmpb_int = \c_one_int
7249        {
7250          \tl_if_empty:NTF \l_@@_hpos_cell_tl
7251            { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7252            { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7253        }
7254        { \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.

```
7255      \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
```

169

```
7256      \tl_set:Ne \l_tmpa_tl
7257        {
7258          { \int_use:N \c@iRow }
7259          { \int_use:N \c@jCol }
7260          { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7261          { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7262        }
```

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).

```
7263      \bool_set_false:N \l_tmpa_bool
7264      \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
7265        { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7266      \bool_case:nF
7267        {
7268          \l_tmpa_bool                               { \@@_Block_vii:eennn }
7269          \l_@@_p_block_bool                         { \@@_Block_vi:eennn }
```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the p, m and b columns and we should modify that point. However, for the X column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```
7270          \l_@@_X_bool                               { \@@_Block_v:eennn }
7271          { \tl_if_empty_p:n { #5 } }                { \@@_Block_v:eennn }
7272          { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7273          { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7274        }
7275        { \@@_Block_v:eennn }
7276      { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7277    }
```

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.

```
7278  \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7279  \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7280    {
7281      \int_gincr:N \g_@@_block_box_int
7282      \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7283        {
7284          \tl_gput_right:Ne \g_@@_pre_code_after_tl
7285            {
7286              \@@_actually_diagbox:nnnnnn
7287                { \int_use:N \c@iRow }
7288                { \int_use:N \c@jCol }
7289                { \int_eval:n { \c@iRow + #1 - 1 } }
7290                { \int_eval:n { \c@jCol + #2 - 1 } }
7291                { \g_@@_row_style_tl \exp_not:n { ##1 } }
7292                { \g_@@_row_style_tl \exp_not:n { ##2 } }
```

```
7293                  }
7294              }
7295          \box_gclear_new:c
7296              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful*: if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```
7297          \hbox_gset:cn
7298              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7299              {
```

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}).

```
7300              \tl_if_empty:NTF \l_@@_color_tl
7301                  { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7302                  { \@@_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.

```
7303              \int_compare:nNnT { #1 } = \c_one_int
7304                  {
7305                      \int_if_zero:nTF \c@iRow
7306                          {
```

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.

```
$\begin{bNiceMatrix}%
  [
    r,
    first-row,
    last-col,
    code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
    code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
  ]
    &   &    &  & \\
 -2 & 3 & -4 & 5 & \\
  3 & -4 & 5 & -6 & \\
 -4 & 5 & -6 & 7 & \\
  5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$
```

```
7307                              \cs_set_eq:NN \Block \@@_NullBlock:
7308                              \l_@@_code_for_first_row_tl
7309                          }
7310                          {
7311                              \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7312                                  {
7313                                      \cs_set_eq:NN \Block \@@_NullBlock:
7314                                      \l_@@_code_for_last_row_tl
7315                                  }
7316                          }
7317                      \g_@@_row_style_tl
7318                  }
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7319            \@@_reset_arraystretch:
7320            \dim_zero:N \extrarowheight
```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```
7321            #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.).

```
7322            \@@_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}`.

```
7323            \bool_if:NTF \l_@@_tabular_bool
7324              {
7325                \bool_lazy_all:nTF
7326                  {
7327                    { \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.

```
7328                    { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7329                    { ! \g_@@_rotate_bool }
7330                  }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7331                  {
7332                    \use:e
7333                      {
```

The `\exp_not:N` is mandatory before `\begin`.

```
7334                        \exp_not:N \begin { minipage }%
7335                          [ \str_lowercase:o \l_@@_vpos_block_str ]
7336                          { \l_@@_col_width_dim }
7337                        \str_case:on \l_@@_hpos_block_str
7338                          { c \centering r \raggedleft l \raggedright }
7339                      }
7340                    #5
7341                    \end { minipage }
7342                  }
```

In the other cases, we use a `{tabular}`.

```
7343                  {
7344                    \bool_if:NT \c_@@_testphase_table_bool
7345                      { \tagpdfsetup { table / tagging = presentation } }
7346                    \use:e
7347                      {
7348                        \exp_not:N \begin { tabular }%
7349                          [ \str_lowercase:o \l_@@_vpos_block_str ]
7350                          { @ { } \l_@@_hpos_block_str @ { } }
7351                      }
7352                    #5
7353                    \end { tabular }
7354                  }
7355              }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7356              {
7357                \c_math_toggle_token
7358                \use:e
7359                  {
```

```
7360              \exp_not:N \begin { array }%
7361                [ \str_lowercase:o \l_@@_vpos_block_str ]
7362                { @ { } \l_@@_hpos_block_str @ { } }
7363            }
7364            #5
7365          \end { array }
7366          \c_math_toggle_token
7367        }
7368      }
```

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).

```
7369      \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.

```
7370      \int_compare:nNnT { #2 } = \c_one_int
7371        {
7372          \dim_gset:Nn \g_@@_blocks_wd_dim
7373            {
7374              \dim_max:nn
7375                \g_@@_blocks_wd_dim
7376                {
7377                  \box_wd:c
7378                    { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7379                }
7380            }
7381        }
```

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 explicitely an option of vertical position.

```
7382      \bool_lazy_and:nnT
7383        { \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.

```
7384        { \str_if_empty_p:N \l_@@_vpos_block_str }
7385        {
7386          \dim_gset:Nn \g_@@_blocks_ht_dim
7387            {
7388              \dim_max:nn
7389                \g_@@_blocks_ht_dim
7390                {
7391                  \box_ht:c
7392                    { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7393                }
7394            }
7395          \dim_gset:Nn \g_@@_blocks_dp_dim
7396            {
7397              \dim_max:nn
7398                \g_@@_blocks_dp_dim
7399                {
7400                  \box_dp:c
7401                    { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7402                }
7403            }
7404        }
7405      \seq_gput_right:Ne \g_@@_blocks_seq
7406        {
7407          \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.

```
7408            {
7409              \exp_not:n { #3 } ,
7410              \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
7411            \bool_if:NT \g_@@_rotate_bool
7412              {
7413                \bool_if:NTF \g_@@_rotate_c_bool
7414                  { m }
7415                  { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7416              }
7417          }
7418          {
7419            \box_use_drop:c
7420              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7421          }
7422        }
7423      \bool_set_false:N \g_@@_rotate_c_bool
7424    }
```

```
7425  \cs_new:Npn \@@_adjust_hpos_rotate:
7426    {
7427      \bool_if:NT \g_@@_rotate_bool
7428        {
7429          \str_set:Ne \l_@@_hpos_block_str
7430            {
7431              \bool_if:NTF \g_@@_rotate_c_bool
7432                { c }
7433                {
7434                  \str_case:onF \l_@@_vpos_block_str
7435                    { b l B l t r T r }
7436                    { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7437                }
7438            }
7439        }
7440    }
```

Despite its name the following command rotates the box of the block *but also does vertical adjustement of the baseline of the block.*

```
7441  \cs_new_protected:Npn \@@_rotate_box_of_block:
7442    {
7443      \box_grotate:cn
7444        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7445        { 90 }
7446      \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7447        {
7448          \vbox_gset_top:cn
7449            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7450            {
7451              \skip_vertical:n { 0.8 ex }
7452              \box_use:c
7453                { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7454            }
7455        }
7456      \bool_if:NT \g_@@_rotate_c_bool
7457        {
7458          \hbox_gset:cn
7459            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
```

```
7460            {
7461              \c_math_toggle_token
7462              \vcenter
7463                {
7464                  \box_use:c
7465                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7466                }
7467              \c_math_toggle_token
7468            }
7469          }
7470      }
```

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:nnnnnn`).

`#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the math mode and before the composition of the block and `#5` is the label (=content) of the block.

```
7471  \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7472  \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7473    {
7474      \seq_gput_right:Ne \g_@@_blocks_seq
7475        {
7476          \l_tmpa_tl
7477          { \exp_not:n { #3 } }
7478          {
7479            \bool_if:NTF \l_@@_tabular_bool
7480              {
7481                \group_begin:
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7482                \@@_reset_arraystretch:
7483                \exp_not:n
7484                  {
7485                    \dim_zero:N \extrarowheight
7486                    #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).

```
7487                    \bool_if:NT \c_@@_testphase_table_bool
7488                      { \tag_stop:n { table } }
7489                    \use:e
7490                      {
7491                        \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7492                        { @ { } \l_@@_hpos_block_str @ { } }
7493                      }
7494                    #5
7495                    \end { tabular }
7496                  }
7497                \group_end:
7498              }
```

When we are *not* in an environment {NiceTabular} (or similar).

```
7499              {
7500                \group_begin:
```

The following will be no-op when `respect-arraystretch` is in force.

```
7501                \@@_reset_arraystretch:
7502                \exp_not:n
7503                  {
```

```
7504              \dim_zero:N \extrarowheight
7505              #4
7506              \c_math_toggle_token
7507              \use:e
7508                {
7509                  \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7510                  { @ { } \l_@@_hpos_block_str @ { } }
7511                }
7512                #5
7513              \end { array }
7514              \c_math_toggle_token
7515            }
7516          \group_end:
7517        }
7518      }
7519    }
7520  }
```

The following macro is for the case of a `\Block` which uses the key p.

```
7521 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7522 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7523  {
7524    \seq_gput_right:Ne \g_@@_blocks_seq
7525      {
7526        \l_tmpa_tl
7527        { \exp_not:n { #3 } }
```

Here, the curly braces for the group are mandatory.

```
7528        { { \exp_not:n { #4 #5 } } } }
7529      }
7530  }
```

The following macro is also for the case of a `\Block` which uses the key p.

```
7531 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7532 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7533  {
7534    \seq_gput_right:Ne \g_@@_blocks_seq
7535      {
7536        \l_tmpa_tl
7537        { \exp_not:n { #3 } }
7538        { \exp_not:n { #4 #5 } }
7539      }
7540  }
```

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).

```
7541 \keys_define:nn { nicematrix / Block / SecondPass }
7542  {
7543    ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7544    ampersand-in-blocks .default:n = true ,
7545    &-in-blocks .meta:n = ampersand-in-blocks ,
```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```
7546    tikz .code:n =
7547      \IfPackageLoadedTF { tikz }
7548        { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7549        { \@@_error:n { tikz~key~without~tikz } } ,
7550    tikz .value_required:n = true ,
7551    fill .code:n =
7552      \tl_set_rescan:Nnn
```

```
7553        \l_@@_fill_tl
7554        { \char_set_catcode_other:N ! }
7555        { #1 } ,
7556      fill .value_required:n = true ,
7557      opacity .tl_set:N = \l_@@_opacity_tl ,
7558      opacity .value_required:n = true ,
7559      draw .code:n =
7560        \tl_set_rescan:Nnn
7561          \l_@@_draw_tl
7562          { \char_set_catcode_other:N ! }
7563          { #1 } ,
7564      draw .default:n = default ,
7565      rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7566      rounded-corners .default:n = 4 pt ,
7567      color .code:n =
7568        \@@_color:n { #1 }
7569        \tl_set_rescan:Nnn
7570          \l_@@_draw_tl
7571          { \char_set_catcode_other:N ! }
7572          { #1 } ,
7573      borders .clist_set:N = \l_@@_borders_clist ,
7574      borders .value_required:n = true ,
7575      hvlines .meta:n = { vlines , hlines } ,
7576      vlines .bool_set:N = \l_@@_vlines_block_bool,
7577      vlines .default:n = true ,
7578      hlines .bool_set:N = \l_@@_hlines_block_bool,
7579      hlines .default:n = true ,
7580      line-width .dim_set:N = \l_@@_line_width_dim ,
7581      line-width .value_required:n = true ,
```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```
7582      j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7583                  \bool_set_true:N \l_@@_p_block_bool ,
7584      l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7585      r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7586      c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7587      L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7588                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7589      R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7590                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7591      C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7592                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7593      t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7594      T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7595      b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7596      B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7597      m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7598      m .value_forbidden:n = true ,
7599      v-center .meta:n = m ,
7600      p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7601      p .value_forbidden:n = true ,
7602      name .tl_set:N = \l_@@_block_name_str ,
7603      name .value_required:n = true ,
7604      name .initial:n = ,
7605      respect-arraystretch .code:n =
7606        \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7607      respect-arraystretch .value_forbidden:n = true ,
7608      transparent .bool_set:N = \l_@@_transparent_bool ,
7609      transparent .default:n = true ,
7610      transparent .initial:n = false ,
7611      unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7612    }
```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construc-

tion of the array. We have to revert to a clean version of \ialign because there may be tabulars in the \Block instructions that will be composed now.

```
7613 \cs_new_protected:Npn \@@_draw_blocks:
7614   {
7615     \bool_if:NTF \c_@@_recent_array_bool
7616       { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7617       { \cs_set_eq:NN \ialign \@@_old_ialign: }
7618     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7619   }

7620 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7621   {
```

The integer \l_@@_last_row_int will be the last row of the block and \l_@@_last_col_int its last column.

```
7622     \int_zero_new:N \l_@@_last_row_int
7623     \int_zero_new:N \l_@@_last_col_int
```

We remind that the first mandatory argument of the command \Block is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in \g_@@_blocks_seq as a number of rows (resp. columns) for the block equal to 100. That's what we detect now (we write 98 for the case the the command \Block has been issued in the "first row").

```
7624     \int_compare:nNnTF { #3 } > { 98 }
7625       { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7626       { \int_set:Nn \l_@@_last_row_int { #3 } }
7627     \int_compare:nNnTF { #4 } > { 98 }
7628       { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7629       { \int_set:Nn \l_@@_last_col_int { #4 } }
7630     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7631       {
7632         \bool_lazy_and:nnTF
7633           \l_@@_preamble_bool
7634           {
7635             \int_compare_p:n
7636              { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7637           }
7638           {
7639             \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7640             \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
7641             \@@_msg_redirect_name:nn { columns~not~used } { none }
7642           }
7643           { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7644       }
7645       {
7646         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7647           { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7648           {
7649             \@@_Block_v:nneenn
7650               { #1 }
7651               { #2 }
7652               { \int_use:N \l_@@_last_row_int }
7653               { \int_use:N \l_@@_last_col_int }
7654               { #5 }
7655               { #6 }
7656           }
7657       }
7658   }
```

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

```
7659    \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7660    \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7661      {
```

The group is for the keys.

```
7662        \group_begin:
7663        \int_compare:nNnT { #1 } = { #3 }
7664          { \str_set:Nn \l_@@_vpos_block_str { t } }
7665        \keys_set:nn { nicematrix / Block / SecondPass } { #5 }
```

If the content of the block contains &, we will have a special treatement (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```
7666        \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7667        \bool_lazy_and:nnT
7668          \l_@@_vlines_block_bool
7669          { ! \l_@@_ampersand_bool }
7670          {
7671            \tl_gput_right:Ne \g_nicematrix_code_after_tl
7672              {
7673                \@@_vlines_block:nnn
7674                  { \exp_not:n { #5 } }
7675                  { #1 - #2 }
7676                  { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7677              }
7678          }
7679        \bool_if:NT \l_@@_hlines_block_bool
7680          {
7681            \tl_gput_right:Ne \g_nicematrix_code_after_tl
7682              {
7683                \@@_hlines_block:nnn
7684                  { \exp_not:n { #5 } }
7685                  { #1 - #2 }
7686                  { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7687              }
7688          }
7689        \bool_if:NF \l_@@_transparent_bool
7690          {
7691            \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7692              {
```

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).

```
7693                \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7694                  { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7695              }
7696          }


7697        \tl_if_empty:NF \l_@@_draw_tl
7698          {
7699            \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7700              { \@@_error:n { hlines~with~color } }
7701            \tl_gput_right:Ne \g_nicematrix_code_after_tl
7702              {
7703                \@@_stroke_block:nnn
```

#5 are the options

```
7704                  { \exp_not:n { #5 } }
7705                  { #1 - #2 }
7706                  { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7707              }
7708            \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7709              { { #1 } { #2 } { #3 } { #4 } }
7710          }
```

```
7711      \clist_if_empty:NF \l_@@_borders_clist
7712        {
7713          \tl_gput_right:Ne \g_nicematrix_code_after_tl
7714            {
7715              \@@_stroke_borders_block:nnn
7716                { \exp_not:n { #5 } }
7717                { #1 - #2 }
7718                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7719            }
7720        }
7721      \tl_if_empty:NF \l_@@_fill_tl
7722        {
7723          \@@_add_opacity_to_fill:
7724          \tl_gput_right:Ne \g_@@_pre_code_before_tl
7725            {
7726              \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7727                { #1 - #2 }
7728                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7729                { \dim_use:N \l_@@_rounded_corners_dim }
7730            }
7731        }
7732      \seq_if_empty:NF \l_@@_tikz_seq
7733        {
7734          \tl_gput_right:Ne \g_nicematrix_code_before_tl
7735            {
7736              \@@_block_tikz:nnnnn
7737                { \seq_use:Nn \l_@@_tikz_seq { , } }
7738                { #1 }
7739                { #2 }
7740                { \int_use:N \l_@@_last_row_int }
7741                { \int_use:N \l_@@_last_col_int }
```

We will have in that last field a list of lists of Tikz keys.

```
7742            }
7743        }

7744      \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7745        {
7746          \tl_gput_right:Ne \g_@@_pre_code_after_tl
7747            {
7748              \@@_actually_diagbox:nnnnnn
7749                { #1 }
7750                { #2 }
7751                { \int_use:N \l_@@_last_row_int }
7752                { \int_use:N \l_@@_last_col_int }
7753                { \exp_not:n { ##1 } }
7754                { \exp_not:n { ##2 } }
7755            }
7756        }
```

Let's consider the following {NiceTabular}. Because of the instruction !{\hspace{1cm}} in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node 1-1-block and the node 1-1-block-short.

```
\begin{NiceTabular}{cc!{\hspace{1cm}}c}
\Block{2-2}{our block} &       & one    \\
                       &       & two    \\
three                  & four  & five   \\
six                    & seven & eight  \\
\end{NiceTabular}
```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```
7757        \pgfpicture
7758        \pgfrememberpicturepositiononpagetrue
7759        \pgf@relevantforpicturesizefalse
7760        \@@_qpoint:n { row - #1 }
7761        \dim_set_eq:NN \l_tmpa_dim \pgf@y
7762        \@@_qpoint:n { col - #2 }
7763        \dim_set_eq:NN \l_tmpb_dim \pgf@x
7764        \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7765        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7766        \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7767        \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.

```
7768        \@@_pgf_rect_node:nnnnn
7769          { \@@_env: - #1 - #2 - block }
7770          \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7771        \str_if_empty:NF \l_@@_block_name_str
7772          {
7773            \pgfnodealias
7774              { \@@_env: - \l_@@_block_name_str }
7775              { \@@_env: - #1 - #2 - block }
7776            \str_if_empty:NF \l_@@_name_str
7777              {
7778                \pgfnodealias
7779                  { \l_@@_name_str - \l_@@_block_name_str }
7780                  { \@@_env: - #1 - #2 - block }
7781              }
7782          }
```

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.

```
7783        \bool_if:NF \l_@@_hpos_of_block_cap_bool
7784          {
7785            \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.

```
7786            \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7787              {
```

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.

```
7788                \cs_if_exist:cT
7789                  { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7790                  {
7791                    \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7792                      {
7793                        \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7794                        \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7795                      }
7796                  }
7797              }
```

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.

```
7798        \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7799          {
7800            \@@_qpoint:n { col - #2 }
7801            \dim_set_eq:NN \l_tmpb_dim \pgf@x
7802          }
7803        \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7804        \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7805          {
7806            \cs_if_exist:cT
7807              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7808              {
7809                \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7810                  {
7811                    \pgfpointanchor
7812                      { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7813                      { east }
7814                    \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7815                  }
7816              }
7817          }
7818        \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7819          {
7820            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7821            \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7822          }
7823        \@@_pgf_rect_node:nnnnn
7824          { \@@_env: - #1 - #2 - block - short }
7825          \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7826      }
```

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.

```
7827      \bool_if:NT \l_@@_medium_nodes_bool
7828        {
7829          \@@_pgf_rect_node:nnn
7830            { \@@_env: - #1 - #2 - block - medium }
7831            { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7832            {
7833              \pgfpointanchor
7834                { \@@_env:
7835                  - \int_use:N \l_@@_last_row_int
7836                  - \int_use:N \l_@@_last_col_int - medium
7837                }
7838                { south~east }
7839            }
7840        }
7841    \endpgfpicture


7842  \bool_if:NTF \l_@@_ampersand_bool
7843    {
7844      \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7845      \int_zero_new:N \l_@@_split_int
7846      \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7847      \pgfpicture
7848      \pgfrememberpicturepositiononpagetrue
7849      \pgf@relevantforpicturesizefalse
7850
7851      \@@_qpoint:n { row - #1 }
7852      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7853      \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
```

```
7854            \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7855            \@@_qpoint:n { col - #2 }
7856            \dim_set_eq:NN \l_tmpa_dim \pgf@x
7857            \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7858            \dim_set:Nn \l_tmpb_dim
7859              { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7860            \bool_lazy_or:nnT
7861              \l_@@_vlines_block_bool
7862              { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7863              {
7864                \int_step_inline:nn { \l_@@_split_int - 1 }
7865                  {
7866                    \pgfpathmoveto
7867                      {
7868                        \pgfpoint
7869                          { \l_tmpa_dim + ##1 \l_tmpb_dim }
7870                          \l_@@_tmpc_dim
7871                      }
7872                    \pgfpathlineto
7873                      {
7874                        \pgfpoint
7875                          { \l_tmpa_dim + ##1 \l_tmpb_dim }
7876                          \l_@@_tmpd_dim
7877                      }
7878                    \CT@arc@
7879                    \pgfsetlinewidth { 1.1 \arrayrulewidth }
7880                    \pgfsetrectcap
7881                    \pgfusepathqstroke
7882                  }
7883              }
7884            \@@_qpoint:n { row - #1 - base }
7885            \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7886            \int_step_inline:nn \l_@@_split_int
7887              {
7888                \group_begin:
7889                \dim_set:Nn \col@sep
7890                  { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7891                \pgftransformshift
7892                  {
7893                    \pgfpoint
7894                      {
7895                        \l_tmpa_dim + ##1 \l_tmpb_dim -
7896                        \str_case:on \l_@@_hpos_block_str
7897                          {
7898                            l { \l_tmpb_dim + \col@sep}
7899                            c { 0.5 \l_tmpb_dim }
7900                            r { \col@sep }
7901                          }
7902                      }
7903                      { \l_@@_tmpc_dim }
7904                  }
7905                \pgfset { inner~sep = \c_zero_dim }
7906                \pgfnode
7907                  { rectangle }
7908                  {
7909                    \str_case:on \l_@@_hpos_block_str
7910                      {
7911                        c { base }
7912                        l { base~west }
7913                        r { base~east }
7914                      }
7915                  }
7916                  { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
```

```
7917            \group_end:
7918          }
7919        \endpgfpicture
7920      }
```

Now the case where there is no ampersand & in the content of the block.

```
7921      {
7922        \bool_if:NTF \l_@@_p_block_bool
7923          {
```

When the final user has used the key p, we have to compute the width.

```
7924            \pgfpicture
7925              \pgfremememberpicturepositiononpagetrue
7926              \pgf@relevantforpicturesizefalse
7927              \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7928                {
7929                  \@@_qpoint:n { col - #2 }
7930                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7931                  \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7932                }
7933                {
7934                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7935                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7936                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7937                }
7938              \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7939            \endpgfpicture
7940            \hbox_set:Nn \l_@@_cell_box
7941              {
7942                \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7943                  { \g_tmpb_dim }
7944                \str_case:on \l_@@_hpos_block_str
7945                  { c \centering r \raggedleft l \raggedright j { } }
7946                #6
7947                \end { minipage }
7948              }
7949          }
7950          { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7951        \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.

```
7952        \pgfpicture
7953        \pgfremememberpicturepositiononpagetrue
7954        \pgf@relevantforpicturesizefalse
7955        \bool_lazy_any:nTF
7956          {
7957            { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7958            { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7959            { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7960            { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7961          }

7962          {
```

If we are in the first column, we must put the block as if it was with the key r.

```
7963            \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.

```
7964            \bool_if:nT \g_@@_last_col_found_bool
7965              {
7966                \int_compare:nNnT { #2 } = \g_@@_col_total_int
7967                  { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7968              }
```

184

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
7969                \tl_set:Ne \l_tmpa_tl
7970                  {
7971                    \str_case:on \l_@@_vpos_block_str
7972                      {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
7973                        { } { % added 2024-06-29
7974                            \str_case:on \l_@@_hpos_block_str
7975                              {
7976                                c { center }
7977                                l { west }
7978                                r { east }
7979                                j { center }
7980                              }
7981                          }
7982                        c {
7983                            \str_case:on \l_@@_hpos_block_str
7984                              {
7985                                c { center }
7986                                l { west }
7987                                r { east }
7988                                j { center }
7989                              }
7990
7991                          }
7992                        T {
7993                            \str_case:on \l_@@_hpos_block_str
7994                              {
7995                                c { north }
7996                                l { north~west }
7997                                r { north~east }
7998                                j { north }
7999                              }
8000
8001                          }
8002                        B {
8003                            \str_case:on \l_@@_hpos_block_str
8004                              {
8005                                c { south }
8006                                l { south~west }
8007                                r { south~east }
8008                                j { south }
8009                              }
8010
8011                          }
8012                      }
8013                  }
8014            \pgftransformshift
8015              {
8016                \pgfpointanchor
8017                  {
8018                    \@@_env: - #1 - #2 - block
8019                    \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8020                  }
8021                  { \l_tmpa_tl }
8022              }
8023            \pgfset { inner~sep = \c_zero_dim }
8024            \pgfnode
8025              { rectangle }
8026              { \l_tmpa_tl }
8027              { \box_use_drop:N \l_@@_cell_box } { } { }
```

```
8028                    }
```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```
8029                    {
8030                      \pgfextracty \l_tmpa_dim
8031                        {
8032                          \@@_qpoint:n
8033                            {
8034                              row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8035                              - base
8036                            }
8037                        }
8038                      \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the x-value of the center of the block.

```
8039                      \pgfpointanchor
8040                        {
8041                          \@@_env: - #1 - #2 - block
8042                          \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8043                        }
8044                        {
8045                          \str_case:on \l_@@_hpos_block_str
8046                            {
8047                              c { center }
8048                              l { west }
8049                              r { east }
8050                              j { center }
8051                            }
8052                        }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
8053                      \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8054                      \pgfset { inner~sep = \c_zero_dim }
8055                      \pgfnode
8056                        { rectangle }
8057                        {
8058                          \str_case:on \l_@@_hpos_block_str
8059                            {
8060                              c { base }
8061                              l { base~west }
8062                              r { base~east }
8063                              j { base }
8064                            }
8065                        }
8066                        { \box_use_drop:N \l_@@_cell_box } { } { }
8067                    }
8068              \endpgfpicture
8069            }
8070          \group_end:
8071      }
```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```
8072  \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8073    {
8074      \pgfpicture
8075      \pgfrememberpicturepositiononpagetrue
8076      \pgf@relevantforpicturesizefalse
8077      \pgfpathrectanglecorners
8078        { \pgfpoint { #2 } { #3 } }
8079        { \pgfpoint { #4 } { #5 } }
8080      \pgfsetfillcolor { #1 }
8081      \pgfusepath { fill }
```

```
8082        \endpgfpicture
8083    }
```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```
8084  \cs_new_protected:Npn \@@_add_opacity_to_fill:
8085    {
8086      \tl_if_empty:NF \l_@@_opacity_tl
8087        {
8088          \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8089            {
8090              \tl_set:Ne \l_@@_fill_tl
8091                {
8092                  [ opacity = \l_@@_opacity_tl ,
8093                  \tl_tail:o \l_@@_fill_tl
8094                }
8095            }
8096            {
8097              \tl_set:Ne \l_@@_fill_tl
8098                { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8099            }
8100        }
8101    }
```

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).

```
8102  \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8103    {
8104      \group_begin:
8105      \tl_clear:N \l_@@_draw_tl
8106      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8107      \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8108      \pgfpicture
8109      \pgfrememberpicturepositiononpagetrue
8110      \pgf@relevantforpicturesizefalse
8111      \tl_if_empty:NF \l_@@_draw_tl
8112        {
```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```
8113          \tl_if_eq:NnTF \l_@@_draw_tl { default }
8114            { \CT@arc@ }
8115            { \@@_color:o \l_@@_draw_tl }
8116        }
8117      \pgfsetcornersarced
8118        {
8119          \pgfpoint
8120          { \l_@@_rounded_corners_dim }
8121          { \l_@@_rounded_corners_dim }
8122        }
8123      \@@_cut_on_hyphen:w #2 \q_stop
8124      \int_compare:nNnF \l_tmpa_tl > \c@iRow
8125        {
8126          \int_compare:nNnF \l_tmpb_tl > \c@jCol
8127            {
8128              \@@_qpoint:n { row - \l_tmpa_tl }
8129              \dim_set_eq:NN \l_tmpb_dim \pgf@y
8130              \@@_qpoint:n { col - \l_tmpb_tl }
8131              \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8132              \@@_cut_on_hyphen:w #3 \q_stop
```

```
8133        \int_compare:nNnT \l_tmpa_tl > \c@iRow
8134          { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8135        \int_compare:nNnT \l_tmpb_tl > \c@jCol
8136          { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8137        \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8138        \dim_set_eq:NN \l_tmpa_dim \pgf@y
8139        \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8140        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8141        \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8142        \pgfpathrectanglecorners
8143          { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8144          { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8145        \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8146          { \pgfusepathqstroke }
8147          { \pgfusepath { stroke } }
8148      }
8149    }
8150    \endpgfpicture
8151    \group_end:
8152  }
```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```
8153  \keys_define:nn { nicematrix / BlockStroke }
8154    {
8155      color .tl_set:N = \l_@@_draw_tl ,
8156      draw .code:n =
8157        \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8158      draw .default:n = default ,
8159      line-width .dim_set:N = \l_@@_line_width_dim ,
8160      rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8161      rounded-corners .default:n = 4 pt
8162    }
```

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).

```
8163  \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8164    {
8165      \group_begin:
8166      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8167      \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8168      \@@_cut_on_hyphen:w #2 \q_stop
8169      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8170      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8171      \@@_cut_on_hyphen:w #3 \q_stop
8172      \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8173      \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8174      \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8175        {
8176          \use:e
8177            {
8178              \@@_vline:n
8179                {
8180                  position = ##1 ,
8181                  start = \l_@@_tmpc_tl ,
8182                  end = \int_eval:n { \l_tmpa_tl - 1 } ,
8183                  total-width = \dim_use:N \l_@@_line_width_dim
8184                }
8185            }
8186        }
8187      \group_end:
8188    }
```

```
8189  \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8190    {
8191      \group_begin:
8192      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8193      \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8194      \@@_cut_on_hyphen:w #2 \q_stop
8195      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8196      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8197      \@@_cut_on_hyphen:w #3 \q_stop
8198      \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8199      \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8200      \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8201        {
8202          \use:e
8203            {
8204              \@@_hline:n
8205                {
8206                  position = ##1 ,
8207                  start = \l_@@_tmpd_tl ,
8208                  end = \int_eval:n { \l_tmpb_tl - 1 } ,
8209                  total-width = \dim_use:N \l_@@_line_width_dim
8210                }
8211            }
8212        }
8213      \group_end:
8214    }
```

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).

```
8215  \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8216    {
8217      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8218      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8219      \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8220        { \@@_error:n { borders~forbidden } }
8221        {
8222          \tl_clear_new:N \l_@@_borders_tikz_tl
8223          \keys_set:no
8224            { nicematrix / OnlyForTikzInBorders }
8225            \l_@@_borders_clist
8226          \@@_cut_on_hyphen:w #2 \q_stop
8227          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8228          \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8229          \@@_cut_on_hyphen:w #3 \q_stop
8230          \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8231          \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8232          \@@_stroke_borders_block_i:
8233        }
8234    }
8235  \hook_gput_code:nnn { begindocument } { . }
8236    {
8237      \cs_new_protected:Npe \@@_stroke_borders_block_i:
8238        {
8239          \c_@@_pgfortikzpicture_tl
8240          \@@_stroke_borders_block_ii:
8241          \c_@@_endpgfortikzpicture_tl
8242        }
8243    }
8244  \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8245    {
8246      \pgfremberpicturepositiononpagetrue
```

```
8247        \pgf@relevantforpicturesizefalse
8248        \CT@arc@
8249        \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8250        \clist_if_in:NnT \l_@@_borders_clist { right }
8251          { \@@_stroke_vertical:n \l_tmpb_tl }
8252        \clist_if_in:NnT \l_@@_borders_clist { left }
8253          { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8254        \clist_if_in:NnT \l_@@_borders_clist { bottom }
8255          { \@@_stroke_horizontal:n \l_tmpa_tl }
8256        \clist_if_in:NnT \l_@@_borders_clist { top }
8257          { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8258      }
8259    \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8260      {
8261        tikz .code:n =
8262          \cs_if_exist:NTF \tikzpicture
8263            { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8264            { \@@_error:n { tikz~in~borders~without~tikz } } ,
8265        tikz .value_required:n = true ,
8266        top .code:n = ,
8267        bottom .code:n = ,
8268        left .code:n = ,
8269        right .code:n = ,
8270        unknown .code:n = \@@_error:n { bad~border }
8271      }
```

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).

```
8272    \cs_new_protected:Npn \@@_stroke_vertical:n #1
8273      {
8274        \@@_qpoint:n \l_@@_tmpc_tl
8275        \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8276        \@@_qpoint:n \l_tmpa_tl
8277        \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8278        \@@_qpoint:n { #1 }
8279        \tl_if_empty:NTF \l_@@_borders_tikz_tl
8280          {
8281            \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8282            \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8283            \pgfusepathqstroke
8284          }
8285          {
8286            \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8287              ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8288          }
8289      }
```

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).

```
8290    \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8291      {
8292        \@@_qpoint:n \l_@@_tmpd_tl
8293        \clist_if_in:NnTF \l_@@_borders_clist { left }
8294          { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8295          { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8296        \@@_qpoint:n \l_tmpb_tl
8297        \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8298        \@@_qpoint:n { #1 }
8299        \tl_if_empty:NTF \l_@@_borders_tikz_tl
8300          {
8301            \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8302            \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8303            \pgfusepathqstroke
```

```
8304        }
8305        {
8306          \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8307            ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8308        }
8309      }
```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```
8310 \keys_define:nn { nicematrix / BlockBorders }
8311   {
8312     borders .clist_set:N = \l_@@_borders_clist ,
8313     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8314     rounded-corners .default:n = 4 pt ,
8315     line-width .dim_set:N = \l_@@_line_width_dim
8316   }
```

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.

```
8317 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8318 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8319   {
8320     \begin { tikzpicture }
8321     \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```
8322     \clist_map_inline:nn { #1 }
8323       {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by nicematrix.

```
8324         \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8325         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8326             (
8327               [
8328                 xshift = \dim_use:N \l_@@_offset_dim ,
8329                 yshift = - \dim_use:N \l_@@_offset_dim
8330               ]
8331               #2 -| #3
8332             )
8333             rectangle
8334             (
8335               [
8336                 xshift = - \dim_use:N \l_@@_offset_dim ,
8337                 yshift = \dim_use:N \l_@@_offset_dim
8338               ]
8339               \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8340             ) ;
8341       }
8342     \end { tikzpicture }
8343   }
```

```
8344 \keys_define:nn { nicematrix / SpecialOffset }
8345   { offset .dim_set:N = \l_@@_offset_dim }
```

In some circonstancies, 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.

```
8346 \cs_new_protected:Npn \@@_NullBlock:
8347   { \@@_collect_options:n { \@@_NullBlock_i: } }
8348 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8349   { }
```

27 How to draw the dotted lines transparently

```
8350 \cs_set_protected:Npn \@@_renew_matrix:
8351   {
8352     \RenewDocumentEnvironment { pmatrix } { }
8353       { \pNiceMatrix }
8354       { \endpNiceMatrix }
8355     \RenewDocumentEnvironment { vmatrix } { }
8356       { \vNiceMatrix }
8357       { \endvNiceMatrix }
8358     \RenewDocumentEnvironment { Vmatrix } { }
8359       { \VNiceMatrix }
8360       { \endVNiceMatrix }
8361     \RenewDocumentEnvironment { bmatrix } { }
8362       { \bNiceMatrix }
8363       { \endbNiceMatrix }
8364     \RenewDocumentEnvironment { Bmatrix } { }
8365       { \BNiceMatrix }
8366       { \endBNiceMatrix }
8367   }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment {NiceArrayWithDelims}.

```
8368 \keys_define:nn { nicematrix / Auto }
8369   {
8370     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8371     columns-type .value_required:n = true ,
8372     l .meta:n = { columns-type = l } ,
8373     r .meta:n = { columns-type = r } ,
8374     c .meta:n = { columns-type = c } ,
8375     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8376     delimiters / color .value_required:n = true ,
8377     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8378     delimiters / max-width .default:n = true ,
8379     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8380     delimiters .value_required:n = true ,
8381     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8382     rounded-corners .default:n = 4 pt
8383   }
8384 \NewDocumentCommand \AutoNiceMatrixWithDelims
8385   { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8386   { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7  } }
8387 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8388   {
```

The group is for the protection of the keys.

```
8389     \group_begin:
8390     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8391     \use:e
8392       {
8393         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8394           { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
```

```
8395              [ \exp_not:o \l_tmpa_tl ]
8396        }
8397      \int_if_zero:nT \l_@@_first_row_int
8398        {
8399          \int_if_zero:nT \l_@@_first_col_int { & }
8400          \prg_replicate:nn { #4 - 1 } { & }
8401          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8402        }
8403      \prg_replicate:nn { #3 }
8404        {
8405          \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`).

```
8406          \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8407          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8408        }
8409      \int_compare:nNnT \l_@@_last_row_int > { -2 }
8410        {
8411          \int_if_zero:nT \l_@@_first_col_int { & }
8412          \prg_replicate:nn { #4 - 1 } { & }
8413          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8414        }
8415      \end { NiceArrayWithDelims }
8416      \group_end:
8417    }
8418  \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8419    {
8420      \cs_set_protected:cpn { #1 AutoNiceMatrix }
8421        {
8422          \bool_gset_true:N \g_@@_delims_bool
8423          \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8424          \AutoNiceMatrixWithDelims { #2 } { #3 }
8425        }
8426    }
8427  \@@_define_com:nnn p ( )
8428  \@@_define_com:nnn b [ ]
8429  \@@_define_com:nnn v | |
8430  \@@_define_com:nnn V \| \|
8431  \@@_define_com:nnn B \{ \}
```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```
8432  \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8433    {
8434      \group_begin:
8435      \bool_gset_false:N \g_@@_delims_bool
8436      \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8437      \group_end:
8438    }
```

29   The redefinition of the command \dotfill

```
8439  \cs_set_eq:NN \@@_old_dotfill \dotfill
8440  \cs_new_protected:Npn \@@_dotfill:
8441    {
```

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}`).

```
8442        \@@_old_dotfill
8443        \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8444      }
```

Now, if the box if not empty (unfornately, 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`.

```
8445 \cs_new_protected:Npn \@@_dotfill_i:
8446   { \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 circonstancies.

```
8447 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8448   {
8449     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8450       {
8451         \@@_actually_diagbox:nnnnnn
8452           { \int_use:N \c@iRow }
8453           { \int_use:N \c@jCol }
8454           { \int_use:N \c@iRow }
8455           { \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 chunck of instructions.

```
8456          { \g_@@_row_style_tl \exp_not:n { #1 } }
8457          { \g_@@_row_style_tl \exp_not:n { #2 } }
8458       }
```

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.

```
8459     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8460       {
8461         { \int_use:N \c@iRow }
8462         { \int_use:N \c@jCol }
8463         { \int_use:N \c@iRow }
8464         { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8465          { }
8466       }
8467   }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```
8468 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8469   {
8470     \pgfpicture
8471     \pgf@relevantforpicturesizefalse
8472     \pgfrememberpicturepositiononpagetrue
8473     \@@_qpoint:n { row - #1 }
8474     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8475     \@@_qpoint:n { col - #2 }
8476     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8477     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
```

```
8478        \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8479        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8480        \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8481        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8482        \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8483        {
```

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.

```
8484          \CT@arc@
8485          \pgfsetroundcap
8486          \pgfusepathqstroke
8487        }
8488        \pgfset { inner~sep = 1 pt }
8489        \pgfscope
8490        \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8491        \pgfnode { rectangle } { south~west }
8492          {
8493            \begin { minipage } { 20 cm }
```

The \scan_stop: avoids an error in math mode when the argument #5 is empty.

```
8494            \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8495            \end { minipage }
8496          }
8497          { }
8498          { }
8499        \endpgfscope
8500        \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8501        \pgfnode { rectangle } { north~east }
8502          {
8503            \begin { minipage } { 20 cm }
8504            \raggedleft
8505            \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8506            \end { minipage }
8507          }
8508          { }
8509          { }
8510        \endpgfpicture
8511      }
```

31 The keyword \CodeAfter

In fact, in this subsection, we define the user command \CodeAfter for the case of the "normal syntax". For the case of "light-syntax", see the definition of the environment {@@-light-syntax} on p. 83.

In the environments of nicematrix, \CodeAfter will be linked to \@@_CodeAfter:. That macro must *not* be protected since it begins with \omit.

```
8512 \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 \\.

```
8513 \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.

```
8514 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8515   {
8516     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
```

```
8517        \@@_CodeAfter_iv:n
8518    }
```

We catch the argument of the command \end (in #1).

```
8519  \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8520    {
```

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.

```
8521      \str_if_eq:eeTF \@currenvir { #1 }
8522        { \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.

```
8523        {
8524          \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8525          \@@_CodeAfter_ii:n
8526        }
8527    }
```

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 colummn. 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.

```
8528  \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8529    {
8530      \pgfpicture
8531      \pgfrememberpicturepositiononpagetrue
8532      \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.

```
8533      \@@_qpoint:n { row - 1 }
8534      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8535      \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8536      \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).

```
8537      \bool_if:nTF { #3 }
8538        { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8539        { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8540      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8541        {
8542          \cs_if_exist:cT
8543            { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8544            {
8545              \pgfpointanchor
8546                { \@@_env: - ##1 - #2 }
8547                { \bool_if:nTF { #3 } { west } { east } }
8548              \dim_set:Nn \l_tmpa_dim
```

```
8549              { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8550          }
8551       }
```
Now we can put the delimiter with a node of PGF.
```
8552       \pgfset { inner~sep = \c_zero_dim }
8553       \dim_zero:N \nulldelimiterspace
8554       \pgftransformshift
8555         {
8556           \pgfpoint
8557             { \l_tmpa_dim }
8558             { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8559         }
8560       \pgfnode
8561         { rectangle }
8562         { \bool_if:nTF { #3 } { east } { west } }
8563         {
```
Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.
```
8564           \nullfont
8565           \c_math_toggle_token
8566           \@@_color:o \l_@@_delimiters_color_tl
8567           \bool_if:nTF { #3 } { \left #1 } { \left . }
8568           \vcenter
8569             {
8570               \nullfont
8571               \hrule \@height
8572                      \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8573                      \@depth \c_zero_dim
8574                      \@width \c_zero_dim
8575             }
8576           \bool_if:nTF { #3 } { \right . } { \right #1 }
8577           \c_math_toggle_token
8578         }
8579         { }
8580         { }
8581       \endpgfpicture
8582   }
```

33 The command \SubMatrix

```
8583 \keys_define:nn { nicematrix / sub-matrix }
8584   {
8585     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8586     extra-height .value_required:n = true ,
8587     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8588     left-xshift .value_required:n = true ,
8589     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8590     right-xshift .value_required:n = true ,
8591     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8592     xshift .value_required:n = true ,
8593     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8594     delimiters / color .value_required:n = true ,
8595     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8596     slim .default:n = true ,
8597     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8598     hlines .default:n = all ,
8599     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8600     vlines .default:n = all ,
8601     hvlines .meta:n = { hlines, vlines } ,
8602     hvlines .value_forbidden:n = true
```

```
8603      }
8604  \keys_define:nn { nicematrix }
8605      {
8606        SubMatrix .inherit:n = nicematrix / sub-matrix ,
8607        NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8608        pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8609        NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8610      }
```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```
8611  \keys_define:nn { nicematrix / SubMatrix }
8612      {
8613        delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8614        delimiters / color .value_required:n = true ,
8615        hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8616        hlines .default:n = all ,
8617        vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8618        vlines .default:n = all ,
8619        hvlines .meta:n = { hlines, vlines } ,
8620        hvlines .value_forbidden:n = true ,
8621        name .code:n =
8622          \tl_if_empty:nTF { #1 }
8623            { \@@_error:n { Invalid~name } }
8624            {
8625              \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8626                {
8627                  \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8628                    { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8629                    {
8630                      \str_set:Nn \l_@@_submatrix_name_str { #1 }
8631                      \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8632                    }
8633                }
8634                { \@@_error:n { Invalid~name } }
8635            } ,
8636        name .value_required:n = true ,
8637        rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8638        rules .value_required:n = true ,
8639        code .tl_set:N = \l_@@_code_tl ,
8640        code .value_required:n = true ,
8641        unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8642      }


8643  \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8644      {
8645        \peek_remove_spaces:n
8646          {
8647            \tl_gput_right:Ne \g_@@_pre_code_after_tl
8648              {
8649                \SubMatrix { #1 } { #2 } { #3 } { #4 }
8650                  [
8651                    delimiters / color = \l_@@_delimiters_color_tl ,
8652                    hlines = \l_@@_submatrix_hlines_clist ,
8653                    vlines = \l_@@_submatrix_vlines_clist ,
8654                    extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8655                    left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8656                    right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8657                    slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8658                    #5
8659                  ]
8660              }
8661            \@@_SubMatrix_in_code_before_i { #2 } { #3 }
```

```
8662            }
8663       }
8664  \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8665    { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8666    { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8667  \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8668    {
8669      \seq_gput_right:Ne \g_@@_submatrix_seq
8670        {
```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```
8671          { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8672          { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8673          { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8674          { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8675        }
8676    }
```

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`.

```
8677  \hook_gput_code:nnn { begindocument } { . }
8678    {
8679      \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m O { } E { _ ^ } { { } { } } }
8680      \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
8681      \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8682        {
8683          \peek_remove_spaces:n
8684            {
8685              \@@_sub_matrix:nnnnnnn
8686                { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8687            }
8688        }
8689    }
```

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).

```
8690  \NewDocumentCommand \@@_compute_i_j:nn
8691    { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8692    { \@@_compute_i_j:nnnn #1 #2 }
8693  \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8694    {
8695      \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8696      \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8697      \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8698      \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8699      \tl_if_eq:NnT \l_@@_first_i_tl { last }
```

```
8700        { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8701      \tl_if_eq:NnT \l_@@_first_j_tl { last }
8702        { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8703      \tl_if_eq:NnT \l_@@_last_i_tl { last }
8704        { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8705      \tl_if_eq:NnT \l_@@_last_j_tl { last }
8706        { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8707    }
8708  \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8709    {
8710      \group_begin:
```

The four following token lists correspond to the position of the \SubMatrix.

```
8711      \@@_compute_i_j:nn { #2 } { #3 }
8712      \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8713        { \cs_set_nopar:Npn \arraystretch { 1 } }
8714      \bool_lazy_or:nnTF
8715        { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8716        { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8717        { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8718        {
8719          \str_clear_new:N \l_@@_submatrix_name_str
8720          \keys_set:nn { nicematrix / SubMatrix } { #5 }
8721          \pgfpicture
8722          \pgfrememberpicturepositiononpagetrue
8723          \pgf@relevantforpicturesizefalse
8724          \pgfset { inner~sep = \c_zero_dim }
8725          \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8726          \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
```

The last value of \int_step_inline:nnn is provided by currifycation.

```
8727          \bool_if:NTF \l_@@_submatrix_slim_bool
8728            { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8729            { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8730            {
8731              \cs_if_exist:cT
8732                { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8733                {
8734                  \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8735                  \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8736                    { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8737                }
8738              \cs_if_exist:cT
8739                { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8740                {
8741                  \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8742                  \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8743                    { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8744                }
8745            }
8746          \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8747            { \@@_error:nn { Impossible~delimiter } { left } }
8748            {
8749              \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8750                { \@@_error:nn { Impossible~delimiter } { right } }
8751                { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8752            }
8753          \endpgfpicture
8754        }
8755      \group_end:
8756    }
```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```
8757 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8758   {
8759     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8760     \dim_set:Nn \l_@@_y_initial_dim
8761       {
8762         \fp_to_dim:n
8763           {
8764             \pgf@y
8765             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8766           }
8767       }
8768     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8769     \dim_set:Nn \l_@@_y_final_dim
8770       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } } }
8771     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8772       {
8773         \cs_if_exist:cT
8774           { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8775           {
8776             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8777             \dim_set:Nn \l_@@_y_initial_dim
8778               { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8779           }
8780         \cs_if_exist:cT
8781           { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8782           {
8783             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8784             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8785               { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8786           }
8787       }
8788     \dim_set:Nn \l_tmpa_dim
8789       {
8790         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8791         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8792       }
8793     \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
8794     \group_begin:
8795     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8796     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8797     \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.

```
8798     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8799       {
8800         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8801           {
8802             \int_compare:nNnT
8803               { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8804               {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
8805                 \@@_qpoint:n { col - ##1 }
8806                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8807                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8808                 \pgfusepathqstroke
8809               }
8810           }
8811       }
```

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.

```
8812        \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8813          { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8814          { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8815          {
8816            \bool_lazy_and:nnTF
8817              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8818              {
8819                 \int_compare_p:nNn
8820                   { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8821              {
8822                \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8823                \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8824                \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8825                \pgfusepathqstroke
8826              }
8827              { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8828          }
```

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.

```
8829        \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8830          { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8831          { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8832          {
8833            \bool_lazy_and:nnTF
8834              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8835              {
8836                \int_compare_p:nNn
8837                  { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8838              {
8839                \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
8840                \group_begin:
```

We compute in `\l_tmpa_dim` the x-value of the left end of the rule.

```
8841                \dim_set:Nn \l_tmpa_dim
8842                  { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8843                \str_case:nn { #1 }
8844                  {
8845                    (  { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8846                    [  { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8847                    \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8848                  }
8849                \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

We compute in `\l_tmpb_dim` the x-value of the right end of the rule.

```
8850                \dim_set:Nn \l_tmpb_dim
8851                  { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8852                \str_case:nn { #2 }
8853                  {
8854                    )  { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8855                    ]  { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8856                    \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8857                  }
8858                \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8859                \pgfusepathqstroke
8860                \group_end:
8861              }
8862              { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8863          }
```

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).

```
8864      \str_if_empty:NF \l_@@_submatrix_name_str
8865        {
8866          \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8867            \l_@@_x_initial_dim \l_@@_y_initial_dim
8868            \l_@@_x_final_dim \l_@@_y_final_dim
8869        }
8870      \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`.

```
8871      \begin { pgfscope }
8872      \pgftransformshift
8873        {
8874          \pgfpoint
8875            { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8876            { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8877        }
8878      \str_if_empty:NTF \l_@@_submatrix_name_str
8879        { \@@_node_left:nn #1 { } }
8880        { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8881      \end { pgfscope }
```

Now, we deal with the right delimiter.

```
8882      \pgftransformshift
8883        {
8884          \pgfpoint
8885            { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8886            { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8887        }
8888      \str_if_empty:NTF \l_@@_submatrix_name_str
8889        { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8890        {
8891          \@@_node_right:nnnn #2
8892            { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8893        }
```

Now, we deal with the key `code` of `\SubMatrix`. That key should contain a TikZ instruction and the nodes in that instruction will be relative to the current `\SubMatrix`. That's why we need a redefinition of `\pgfpointanchor`.

```
8894      \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8895      \flag_clear_new:N \l_@@_code_flag
8896      \l_@@_code_tl
8897    }
```

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`.

```
8898 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor
```

The following command will be linked to `\pgfpointanchor` just before the execution of the option `code` of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
8899 \cs_new:Npn \@@_pgfpointanchor:n #1
8900   { \exp_args:Ne \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`.

```
8901 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8902   { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
```

```
8903 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8904   {
```

The command `\str_if_empty:nTF` is "fully expandable".

```
8905     \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
8906       { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
8907       { \@@_pgfpointanchor_ii:n { #1 } }
8908   }
```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```
8909 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8910   { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package etl but, as of now, we do not load etl.

```
8911 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1-\q_stop }
```

```
8912 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2\q_stop
8913   {
```

The command `\str_if_empty:nTF` is "fully expandable".

```
8914     \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8915       { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retreive the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
8916       { \@@_pgfpointanchor_iii:w { #1 } #2 }
8917   }
```

The following function is for the case when the name contains an hyphen.

```
8918 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8919   {
```

We have to add the prefix `\@@_env:` "by hand" since we have retreived the potential `\tikz@pp@name`.

```
8920     \@@_env:
8921     - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8922     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8923   }
```

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.

```
8924 \tl_const:Nn \c_@@_integers_alist_tl
8925   {
8926     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8927     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8928     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8929     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8930   }
```

```
8931 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8932   {
```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form i-$|j$. That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises witht its command `\name_of_command` (see above).

In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```
8933       \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8934         {
8935           \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env:` "by hand" since we have retreived the potential `\tikz@pp@name`.

```
8936           \@@_env: -
8937           \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8938             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8939             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8940         }
8941         {
8942           \str_if_eq:eeTF { #1 } { last }
8943             {
8944               \flag_raise:N \l_@@_code_flag
8945               \@@_env: -
8946               \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8947                 { \int_eval:n { \l_@@_last_i_tl + 1 } }
8948                 { \int_eval:n { \l_@@_last_j_tl + 1 } }
8949             }
8950             { #1 }
8951         }
8952   }
```

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`).

```
8953 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8954   {
8955     \pgfnode
8956       { rectangle }
8957       { east }
8958       {
8959         \nullfont
8960         \c_math_toggle_token
8961         \@@_color:o \l_@@_delimiters_color_tl
8962         \left #1
8963         \vcenter
8964           {
8965             \nullfont
8966             \hrule \@height \l_tmpa_dim
8967                   \@depth \c_zero_dim
```

205

```
8968                     \@width \c_zero_dim
8969                 }
8970             \right .
8971             \c_math_toggle_token
8972         }
8973       { #2 }
8974       { }
8975   }
```

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.

```
8976 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8977   {
8978     \pgfnode
8979       { rectangle }
8980       { west }
8981       {
8982         \nullfont
8983         \c_math_toggle_token
8984         \colorlet { current-color } { . }
8985         \@@_color:o \l_@@_delimiters_color_tl
8986         \left .
8987         \vcenter
8988           {
8989             \nullfont
8990             \hrule \@height \l_tmpa_dim
8991                   \@depth \c_zero_dim
8992                   \@width \c_zero_dim
8993           }
8994         \right #1
8995         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8996         ^ { \color { current-color } \smash { #4 } }
8997         \c_math_toggle_token
8998       }
8999       { #2 }
9000       { }
9001   }
```

34 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```
9002 \NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
9003   {
9004     \peek_remove_spaces:n
9005       { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
9006   }
9007 \NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
9008   {
9009     \peek_remove_spaces:n
9010       { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
9011   }

9012 \keys_define:nn { nicematrix / Brace }
9013   {
9014     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9015     left-shorten .default:n = true ,
9016     left-shorten .value_forbidden:n = true ,
```

```
9017      right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9018      right-shorten .default:n = true ,
9019      right-shorten .value_forbidden:n = true ,
9020      shorten .meta:n = { left-shorten , right-shorten } ,
9021      shorten .value_forbidden:n = true ,
9022      yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9023      yshift .value_required:n = true ,
9024      yshift .initial:n = \c_zero_dim ,
9025      color .tl_set:N = \l_tmpa_tl ,
9026      color .value_required:n = true ,
9027      unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9028    }
```

#1 is the first cell of the rectangle (with the syntax i-$|j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to under or over.

```
9029    \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9030      {
9031        \group_begin:
```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```
9032        \@@_compute_i_j:nn { #1 } { #2 }
9033        \bool_lazy_or:nnTF
9034          { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9035          { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9036          {
9037            \str_if_eq:eeTF { #5 } { under }
9038              { \@@_error:nn { Construct~too~large } { \UnderBrace } }
9039              { \@@_error:nn { Construct~too~large } { \OverBrace } }
9040          }
9041          {
9042            \tl_clear:N \l_tmpa_tl
9043            \keys_set:nn { nicematrix / Brace } { #4 }
9044            \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9045            \pgfpicture
9046            \pgfrememberpicturepositiononpagetrue
9047            \pgf@relevantforpicturesizefalse
9048            \bool_if:NT \l_@@_brace_left_shorten_bool
9049              {
9050                \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9051                \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9052                  {
9053                    \cs_if_exist:cT
9054                      { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9055                      {
9056                        \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }

9058                        \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9059                          { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9060                      }
9061                  }
9062              }
9063            \bool_lazy_or:nnT
9064              { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9065              { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9066              {
9067                \@@_qpoint:n { col - \l_@@_first_j_tl }
9068                \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9069              }
9070            \bool_if:NT \l_@@_brace_right_shorten_bool
9071              {
9072                \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9073                \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9074                  {
```

```
9075              \cs_if_exist:cT
9076                { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9077                {
9078                  \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9079                  \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9080                    { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9081                }
9082            }
9083          }
9084        \bool_lazy_or:nnT
9085          { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9086          { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9087          {
9088            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9089            \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9090          }
9091        \pgfset { inner~sep = \c_zero_dim }
9092        \str_if_eq:eeTF { #5 } { under }
9093          { \@@_underbrace_i:n { #3 } }
9094          { \@@_overbrace_i:n { #3 } }
9095        \endpgfpicture
9096      }
9097    \group_end:
9098  }
```

The argument is the text to put above the brace.

```
9099 \cs_new_protected:Npn \@@_overbrace_i:n #1
9100   {
9101     \@@_qpoint:n { row - \l_@@_first_i_tl }
9102     \pgftransformshift
9103       {
9104         \pgfpoint
9105           { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
9106           { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
9107       }
9108     \pgfnode
9109       { rectangle }
9110       { south }
9111       {
9112         \vtop
9113           {
9114             \group_begin:
9115             \everycr { }
9116             \halign
9117               {
9118                 \hfil ## \hfil \crcr
9119                 \bool_if:NTF \l_@@_tabular_bool
9120                   { \begin { tabular } { c } #1 \end { tabular } }
9121                   { $ \begin { array } { c } #1 \end { array } $ }
9122                 \cr
9123                 \c_math_toggle_token
9124                 \overbrace
9125                   {
9126                     \hbox_to_wd:nn
9127                       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9128                       { }
9129                   }
9130                 \c_math_toggle_token
9131               \cr
9132               }
9133             \group_end:
9134           }
9135       }
9136       { }
```

```
9137          { }
9138     }
```

The argument is the text to put under the brace.

```
9139  \cs_new_protected:Npn \@@_underbrace_i:n #1
9140     {
9141       \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9142       \pgftransformshift
9143         {
9144           \pgfpoint
9145             { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
9146             { \pgf@y  - \l_@@_brace_yshift_dim + 3 pt }
9147         }
9148       \pgfnode
9149         { rectangle }
9150         { north }
9151         {
9152           \group_begin:
9153           \everycr { }
9154           \vbox
9155             {
9156               \halign
9157                 {
9158                   \hfil ## \hfil \crcr
9159                   \c_math_toggle_token
9160                   \underbrace
9161                     {
9162                       \hbox_to_wd:nn
9163                         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9164                         { }
9165                     }
9166                   \c_math_toggle_token
9167                   \cr
9168                   \bool_if:NTF \l_@@_tabular_bool
9169                     { \begin { tabular } { c } #1 \end { tabular } }
9170                     { $ \begin { array } { c } #1 \end { array } $ }
9171                   \cr
9172                 }
9173             }
9174           \group_end:
9175         }
9176         { }
9177         { }
9178     }
```

35   The commands HBrace et VBrace

```
9179  \hook_gput_code:nnn { begindocument } { . }
9180     {
9181       \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9182         {
9183           \tikzset
9184             {
9185               nicematrix / brace / .style =
9186                 {
9187                   decoration = { brace , raise = -0.15 em } ,
9188                   decorate ,
9189                 } ,
```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```
9190                nicematrix / mirrored-brace / .style =
9191                  {
9192                    nicematrix / brace ,
9193                    decoration = mirror ,
9194                  }
9195              }
9196          }
9197      }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```
9198  \keys_define:nn { nicematrix / Hbrace }
9199    {
9200      color .code:n = ,
9201      horizontal-labels .code:n = ,
9202      shorten .code:n = ,
9203      shorten-start .code:n = ,
9204      shorten-end .code:n = ,
9205      unknown .code:n = \@@_error:n { Unknown~key~for~Hbrace }
9206    }
```

Here we need an "fully expandable" command.

```
9207  \NewExpandableDocumentCommand { \@@_Hbrace } { O { } m m }
9208    {
9209      \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9210        { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9211        { \@@_error:n { Hbrace~not~allowed } }
9212    }
```

The following command must *not* be protected.

```
9213  \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9214    {
9215      \int_compare:nNnTF \c@iRow < 1
9216        {
```

We recall that `\str_if_eq:nnTF` is "fully expandable".

```
9217          \str_if_eq:nnTF { #2 } { * }
9218            {
9219              \NiceMatrixOptions{nullify-dots}
9220              \Ldots
9221                [
9222                  line-style = nicematrix / brace ,
9223                  #1 ,
9224                  up =
9225                    \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9226                ]
9227            }
9228            {
9229              \Hdotsfor
9230                [
9231                  line-style = nicematrix / brace ,
9232                  #1 ,
9233                  up =
9234                    \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9235                ]
9236                { #2 }
9237            }
9238        }
9239        {
9240          \str_if_eq:nnTF { #2 } { * }
9241            {
9242              \NiceMatrixOptions{nullify-dots}
```

```
9243            \Ldots
9244              [
9245                line-style = nicematrix / mirrored-brace ,
9246                #1 ,
9247                down =
9248                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9249              ]
9250          }
9251          {
9252            \Hdotsfor
9253              [
9254                line-style = nicematrix / mirrored-brace ,
9255                #1 ,
9256                down =
9257                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9258              ]
9259            { #2 }
9260          }
9261      }
9262    \keys_set:nn { nicematrix / Hbrace } { #1 }
9263  }
```

Here we need an "fully expandable" command.

```
9264  \NewExpandableDocumentCommand { \@@_Vbrace } { O { } m m }
9265    {
9266      \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9267        { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9268        { \@@_error:n { Vbrace~not~allowed } }
9269    }
```

The following command must *not* be protected.

```
9270  \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9271    {
9272      \int_compare:nNnTF \c@jCol = 0
9273        {
9274          \str_if_eq:nnTF { #2 } { * }
9275            {
9276              \NiceMatrixOptions{nullify-dots}
9277              \Vdots
9278                [
9279                  line-style = nicematrix / mirrored-brace ,
9280                  #1 ,
9281                  down =
9282                    \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9283                ]
9284            }
9285            {
9286              \Vdotsfor
9287                [
9288                  line-style = nicematrix / mirrored-brace ,
9289                  #1 ,
9290                  down =
9291                    \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9292                ]
9293              { #2 }
9294            }
9295        }
9296        {
9297          \str_if_eq:nnTF { #2 } { * }
9298            {
9299              \NiceMatrixOptions{nullify-dots}
9300              \Vdots
9301                [
9302                  line-style =  nicematrix / brace ,
```

```
9303                  #1 ,
9304                  up =
9305                    \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9306                ]
9307            }
9308            {
9309              \Vdotsfor
9310              [
9311                line-style = nicematrix / brace ,
9312                #1 ,
9313                up =
9314                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9315              ]
9316              { #2 }
9317            }
9318        }
9319    \keys_set:nn { nicematrix / Hbrace } { #1 }
9320  }
```

36 The command TikzEveryCell

```
9321 \bool_new:N \l_@@_not_empty_bool
9322 \bool_new:N \l_@@_empty_bool
9323
9324 \keys_define:nn { nicematrix / TikzEveryCell }
9325  {
9326    not-empty .code:n =
9327      \bool_lazy_or:nnTF
9328        \l_@@_in_code_after_bool
9329        \g_@@_recreate_cell_nodes_bool
9330        { \bool_set_true:N \l_@@_not_empty_bool }
9331        { \@@_error:n { detection~of~empty~cells } } ,
9332    not-empty .value_forbidden:n = true ,
9333    empty .code:n =
9334      \bool_lazy_or:nnTF
9335        \l_@@_in_code_after_bool
9336        \g_@@_recreate_cell_nodes_bool
9337        { \bool_set_true:N \l_@@_empty_bool }
9338        { \@@_error:n { detection~of~empty~cells } } ,
9339    empty .value_forbidden:n = true ,
9340    unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9341  }
9342
9343
9344 \NewDocumentCommand { \@@_TikzEveryCell } { O { } m }
9345  {
9346    \IfPackageLoadedTF { tikz }
9347      {
9348        \group_begin:
9349        \keys_set:nn { nicematrix / TikzEveryCell } { #1 }
```

The inner pair of braces in the following line is mandatory because, the last argument of \@@_tikz:nnnnn is *a list of lists* of TikZ keys.

```
9350        \tl_set:Nn \l_tmpa_tl { { #2 } }
9351        \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9352          { \@@_for_a_block:nnnnn ##1 }
9353        \@@_all_the_cells:
9354        \group_end:
9355      }
9356      { \@@_error:n { TikzEveryCell~without~tikz } }
9357  }
9358
```

```
9359  \tl_new:N \@@_i_tl
9360  \tl_new:N \@@_j_tl
9361
9362
9363  \cs_new_protected:Nn \@@_all_the_cells:
9364    {
9365      \int_step_variable:nNn \c@iRow \@@_i_tl
9366        {
9367          \int_step_variable:nNn \c@jCol \@@_j_tl
9368            {
9369              \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9370                {
9371                  \clist_if_in:NeF \l_@@_corners_cells_clist
9372                    { \@@_i_tl - \@@_j_tl }
9373                    {
9374                      \bool_set_false:N \l_tmpa_bool
9375                      \cs_if_exist:cTF
9376                        { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9377                        {
9378                          \bool_if:NF \l_@@_empty_bool
9379                            { \bool_set_true:N \l_tmpa_bool }
9380                        }
9381                        {
9382                          \bool_if:NF \l_@@_not_empty_bool
9383                            { \bool_set_true:N \l_tmpa_bool }
9384                        }
9385                      \bool_if:NT \l_tmpa_bool
9386                        {
9387                          \@@_block_tikz:onnnn
9388                          \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9389                        }
9390                    }
9391                }
9392            }
9393        }
9394    }
9395
9396  \cs_new_protected:Nn \@@_for_a_block:nnnnn
9397    {
9398      \bool_if:NF \l_@@_empty_bool
9399        {
9400          \@@_block_tikz:onnnn
9401          \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9402        }
9403      \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9404    }
9405
9406  \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9407    {
9408      \int_step_inline:nnn { #1 } { #3 }
9409        {
9410          \int_step_inline:nnn { #2 } { #4 }
9411            { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9412        }
9413    }
```

37   The command \ShowCellNames

```
9414  \NewDocumentCommand \@@_ShowCellNames { }
9415    {
9416      \bool_if:NT \l_@@_in_code_after_bool
9417        {
```

```
9418          \pgfpicture
9419          \pgfrememberpicturepositiononpagetrue
9420          \pgf@relevantforpicturesizefalse
9421          \pgfpathrectanglecorners
9422            { \@@_qpoint:n { 1 } }
9423            {
9424              \@@_qpoint:n
9425                { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9426            }
9427          \pgfsetfillopacity { 0.75 }
9428          \pgfsetfillcolor { white }
9429          \pgfusepathqfill
9430          \endpgfpicture
9431        }
9432      \dim_gzero_new:N \g_@@_tmpc_dim
9433      \dim_gzero_new:N \g_@@_tmpd_dim
9434      \dim_gzero_new:N \g_@@_tmpe_dim
9435      \int_step_inline:nn \c@iRow
9436        {
9437          \bool_if:NTF \l_@@_in_code_after_bool
9438            {
9439              \pgfpicture
9440              \pgfrememberpicturepositiononpagetrue
9441              \pgf@relevantforpicturesizefalse
9442            }
9443            { \begin { pgfpicture } }
9444          \@@_qpoint:n { row - ##1 }
9445          \dim_set_eq:NN \l_tmpa_dim \pgf@y
9446          \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9447          \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9448          \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9449          \bool_if:NTF \l_@@_in_code_after_bool
9450            { \endpgfpicture }
9451            { \end { pgfpicture } }
9452          \int_step_inline:nn \c@jCol
9453            {
9454              \hbox_set:Nn \l_tmpa_box
9455                {
9456                  \normalfont \Large \sffamily \bfseries
9457                  \bool_if:NTF \l_@@_in_code_after_bool
9458                    { \color { red } }
9459                    { \color { red ! 50 } }
9460                  ##1 - ####1
9461                }
9462              \bool_if:NTF \l_@@_in_code_after_bool
9463                {
9464                  \pgfpicture
9465                  \pgfrememberpicturepositiononpagetrue
9466                  \pgf@relevantforpicturesizefalse
9467                }
9468                { \begin { pgfpicture } }
9469              \@@_qpoint:n { col - ####1 }
9470              \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9471              \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9472              \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9473              \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9474              \bool_if:NTF \l_@@_in_code_after_bool
9475                { \endpgfpicture }
9476                { \end { pgfpicture } }
9477              \fp_set:Nn \l_tmpa_fp
9478                {
9479                  \fp_min:nn
9480                    {
```

```
9481                \fp_min:nn
9482                  { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9483                  { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } } }
9484              }
9485            { 1.0 }
9486          }
9487        \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9488        \pgfpicture
9489        \pgfrememberpicturepositiononpagetrue
9490        \pgf@relevantforpicturesizefalse
9491        \pgftransformshift
9492          {
9493            \pgfpoint
9494              { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9495              { \dim_use:N \g_tmpa_dim }
9496          }
9497        \pgfnode
9498          { rectangle }
9499          { center }
9500          { \box_use:N \l_tmpa_box }
9501          { }
9502          { }
9503        \endpgfpicture
9504      }
9505    }
9506 }
```

38 We process the options at package loading

We process the options when the package is loaded (with \usepackage) but we recommend to use
\NiceMatrixOptions instead.

We must process these options after the definition of the environment {NiceMatrix} because the
option renew-matrix executes the code \cs_set_eq:NN \env@matrix \NiceMatrix.

Of course, the command \NiceMatrix must be defined before such an instruction is executed.

The boolean \g_@@_footnotehyper_bool will indicate if the option footnotehyper is used.

```
9507 \bool_new:N \g_@@_footnotehyper_bool
```

The boolean \g_@@_footnote_bool will indicate if the option footnote is used, but quicky, it will
also be set to true if the option footnotehyper is used.

```
9508 \bool_new:N \g_@@_footnote_bool
9509 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9510   {
9511     The~key~'\l_keys_key_str'~is~unknown. \\
9512     That~key~will~be~ignored. \\
9513     For~a~list~of~the~available~keys,~type~H~<return>.
9514   }
9515   {
9516     The~available~keys~are~(in~alphabetic~order):~
9517     footnote,~
9518     footnotehyper,~
9519     messages-for-Overleaf,~
9520     renew-dots,~and~
9521     renew-matrix.
9522   }
9523 \keys_define:nn { nicematrix / Package }
9524   {
9525     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9526     renew-dots .value_forbidden:n = true ,
9527     renew-matrix .code:n = \@@_renew_matrix: ,
```

```
9528        renew-matrix .value_forbidden:n = true ,
9529        messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9530        footnote .bool_set:N = \g_@@_footnote_bool ,
9531        footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9532        unknown .code:n = \@@_error:n { Unknown~key~for~package }
9533      }
9534  \ProcessKeysOptions { nicematrix / Package }
```

```
9535  \@@_msg_new:nn { footnote~with~footnotehyper~package }
9536    {
9537      You~can't~use~the~option~'footnote'~because~the~package~
9538      footnotehyper~has~already~been~loaded.~
9539      If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9540      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9541      of~the~package~footnotehyper.\\
9542      The~package~footnote~won't~be~loaded.
9543    }
9544  \@@_msg_new:nn { footnotehyper~with~footnote~package }
9545    {
9546      You~can't~use~the~option~'footnotehyper'~because~the~package~
9547      footnote~has~already~been~loaded.~
9548      If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9549      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9550      of~the~package~footnote.\\
9551      The~package~footnotehyper~won't~be~loaded.
9552    }
```

```
9553  \bool_if:NT \g_@@_footnote_bool
9554    {
```

The class **beamer** has its own system to extract footnotes and that's why we have nothing to do if **beamer** is used.

```
9555      \IfClassLoadedTF { beamer }
9556        { \bool_set_false:N \g_@@_footnote_bool }
9557        {
9558          \IfPackageLoadedTF { footnotehyper }
9559            { \@@_error:n { footnote~with~footnotehyper~package } }
9560            { \usepackage { footnote } }
9561        }
9562    }
9563  \bool_if:NT \g_@@_footnotehyper_bool
9564    {
```

The class **beamer** has its own system to extract footnotes and that's why we have nothing to do if **beamer** is used.

```
9565      \IfClassLoadedTF { beamer }
9566        { \bool_set_false:N \g_@@_footnote_bool }
9567        {
9568          \IfPackageLoadedTF { footnote }
9569            { \@@_error:n { footnotehyper~with~footnote~package } }
9570            { \usepackage { footnotehyper } }
9571        }
9572      \bool_set_true:N \g_@@_footnote_bool
9573    }
```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment {savenotes}.

39 About the package underscore

If the user loads the package underscore, it must be loaded *before* the package nicematrix. If it is loaded after, we raise an error.

```
9574 \bool_new:N \l_@@_underscore_loaded_bool
9575 \IfPackageLoadedT { underscore }
9576   { \bool_set_true:N \l_@@_underscore_loaded_bool }

9577 \hook_gput_code:nnn { begindocument } { . }
9578   {
9579     \bool_if:NF \l_@@_underscore_loaded_bool
9580       {
9581         \IfPackageLoadedT { underscore }
9582           { \@@_error:n { underscore~after~nicematrix } }
9583       }
9584   }
```

40 Error messages of the package

```
9585 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9586   { \str_const:Nn \c_@@_available_keys_str { } }
9587   {
9588     \str_const:Nn \c_@@_available_keys_str
9589       { For~a~list~of~the~available~keys,~type~H~<return>. }
9590   }
9591 \seq_new:N \g_@@_types_of_matrix_seq
9592 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9593   {
9594     NiceMatrix ,
9595     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9596   }
9597 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9598   { \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`.

```
9599 \cs_new_protected:Npn \@@_error_too_much_cols:
9600   {
9601     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9602       { \@@_fatal:nn { too~much~cols~for~array } }
9603     \int_compare:nNnT \l_@@_last_col_int = { -2 }
9604       { \@@_fatal:n { too~much~cols~for~matrix } }
9605     \int_compare:nNnT \l_@@_last_col_int = { -1 }
9606       { \@@_fatal:n { too~much~cols~for~matrix } }
9607     \bool_if:NF \l_@@_last_col_without_value_bool
9608       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9609   }
```

The following command must *not* be protected since it's used in an error message.

```
9610 \cs_new:Npn \@@_message_hdotsfor:
9611   {
9612     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9613       { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9614   }
9615 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
```

```
9616    {
9617      Incompatible~options.\\
9618      You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9619      The~output~will~not~be~reliable.
9620    }
9621  \@@_msg_new:nn { key~color-inside }
9622    {
9623      Key~deprecated.\\
9624      The~key~'color-inside'~(and~its~alias~'colortbl-like')~is~now~point-less~
9625      and~have~been~deprecated.\\
9626      You~won't~have~similar~message~till~the~end~of~the~document.
9627    }
9628  \@@_msg_new:nn { negative~weight }
9629    {
9630      Negative~weight.\\
9631      The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9632      the~value~'\int_use:N \l_@@_weight_int'.\\
9633      The~absolute~value~will~be~used.
9634    }
9635  \@@_msg_new:nn { last~col~not~used }
9636    {
9637      Column~not~used.\\
9638      The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9639      in~your~\@@_full_name_env:.~However,~you~can~go~on.
9640    }
9641  \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9642    {
9643      Too~much~columns.\\
9644      In~the~row~\int_eval:n { \c@iRow },~
9645      you~try~to~use~more~columns~
9646      than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9647      The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9648      (plus~the~exterior~columns).~This~error~is~fatal.
9649    }
9650  \@@_msg_new:nn { too~much~cols~for~matrix }
9651    {
9652      Too~much~columns.\\
9653      In~the~row~\int_eval:n { \c@iRow },~
9654      you~try~to~use~more~columns~than~allowed~by~your~
9655      \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9656      number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9657      columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9658      Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9659      \token_to_str:N \setcounter\ to~change~that~value).~
9660      This~error~is~fatal.
9661    }

9662  \@@_msg_new:nn { too~much~cols~for~array }
9663    {
9664      Too~much~columns.\\
9665      In~the~row~\int_eval:n { \c@iRow },~
9666      ~you~try~to~use~more~columns~than~allowed~by~your~
9667      \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9668      \int_use:N \g_@@_static_num_of_col_int\
9669      \bool_if:nT
9670        { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9671        { ~(plus~the~exterior~ones) }
9672      since~the~preamble~is~'\g_@@_user_preamble_tl'.\\
9673      This~error~is~fatal.
9674    }

9675  \@@_msg_new:nn { columns~not~used }
```

```
9676      {
9677        Columns~not~used.\\
9678        The~preamble~of~your~\@@_full_name_env:\ is~'\g_@@_user_preamble_tl'.~
9679        It~announces~\int_use:N
9680        \g_@@_static_num_of_col_int\ columns~but~you~only~used~\int_use:N \c@jCol.\\
9681        The~columns~you~did~not~used~won't~be~created.\\
9682        You~won't~have~similar~warning~till~the~end~of~the~document.
9683      }
9684    \@@_msg_new:nn { empty~preamble }
9685      {
9686        Empty~preamble.\\
9687        The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9688        This~error~is~fatal.
9689      }
9690    \@@_msg_new:nn { in~first~col }
9691      {
9692        Erroneous~use.\\
9693        You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9694        That~command~will~be~ignored.
9695      }
9696    \@@_msg_new:nn { in~last~col }
9697      {
9698        Erroneous~use.\\
9699        You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9700        That~command~will~be~ignored.
9701      }
9702    \@@_msg_new:nn { in~first~row }
9703      {
9704        Erroneous~use.\\
9705        You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9706        That~command~will~be~ignored.
9707      }
9708    \@@_msg_new:nn { in~last~row }
9709      {
9710        Erroneous~use.\\
9711        You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9712        That~command~will~be~ignored.
9713      }
9714    \@@_msg_new:nn { TopRule~without~booktabs }
9715      {
9716        Erroneous~use.\\
9717        You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9718        That~command~will~be~ignored.
9719      }
9720    \@@_msg_new:nn { TopRule~without~tikz }
9721      {
9722        Erroneous~use.\\
9723        You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9724        That~command~will~be~ignored.
9725      }
9726    \@@_msg_new:nn { caption~outside~float }
9727      {
9728        Key~caption~forbidden.\\
9729        You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9730        environment.~This~key~will~be~ignored.
9731      }
9732    \@@_msg_new:nn { short-caption~without~caption }
9733      {
9734        You~should~not~use~the~key~'short-caption'~without~'caption'.~
9735        However,~your~'short-caption'~will~be~used~as~'caption'.
```

```
9736     }
9737   \@@_msg_new:nn { double~closing~delimiter }
9738     {
9739       Double~delimiter.\\
9740       You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9741       delimiter.~This~delimiter~will~be~ignored.
9742     }
9743   \@@_msg_new:nn { delimiter~after~opening }
9744     {
9745       Double~delimiter.\\
9746       You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9747       delimiter.~That~delimiter~will~be~ignored.
9748     }
9749   \@@_msg_new:nn { bad~option~for~line-style }
9750     {
9751       Bad~line~style.\\
9752       Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9753       is~'standard'.~That~key~will~be~ignored.
9754     }
9755   \@@_msg_new:nn { corners~with~no-cell-nodes }
9756     {
9757       Incompatible~keys.\\
9758       You~can't~use~the~key~'corners'~here~because~the~key~'no-cell-nodes'~
9759       is~in~force.\\
9760       If~you~go~on,~that~key~will~be~ignored.
9761     }
9762   \@@_msg_new:nn { extra-nodes~with~no-cell-nodes }
9763     {
9764       Incompatible~keys.\\
9765       You~can't~create~'extra~nodes'~here~because~the~key~'no-cell-nodes'~
9766       is~in~force.\\
9767       If~you~go~on,~those~extra~nodes~won't~be~created.
9768     }
9769   \@@_msg_new:nn { Identical~notes~in~caption }
9770     {
9771       Identical~tabular~notes.\\
9772       You~can't~put~several~notes~with~the~same~content~in~
9773       \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9774       If~you~go~on,~the~output~will~probably~be~erroneous.
9775     }
9776   \@@_msg_new:nn { tabularnote~below~the~tabular }
9777     {
9778       \token_to_str:N \tabularnote\ forbidden\\
9779       You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9780       of~your~tabular~because~the~caption~will~be~composed~below~
9781       the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9782       key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
9783       Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9784       no~similar~error~will~raised~in~this~document.
9785     }
9786   \@@_msg_new:nn { Unknown~key~for~rules }
9787     {
9788       Unknown~key.\\
9789       There~is~only~two~keys~available~here:~width~and~color.\\
9790       Your~key~'\l_keys_key_str'~will~be~ignored.
9791     }
9792   \@@_msg_new:nn { Unknown~key~for~Hbrace }
9793     {
9794       Unknown~key.\\
9795       You~have~used~the~key~'\l_keys_key_str'~but~the~only~
```

```
9796        keys~allowed~for~the~commands~\token_to_str:N \Hbrace\
9797        and~\token_to_str:N \Vbrace\ are:~'color',~
9798        'horizontal-labels',~'shorten'~'shorten-end'~
9799        and~'shorten-start'.
9800      }
9801  \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9802      {
9803        Unknown~key.\\
9804        There~is~only~two~keys~available~here:~
9805        'empty'~and~'not-empty'.\\
9806        Your~key~'\l_keys_key_str'~will~be~ignored.
9807      }
9808  \@@_msg_new:nn { Unknown~key~for~rotate }
9809      {
9810        Unknown~key.\\
9811        The~only~key~available~here~is~'c'.\\
9812        Your~key~'\l_keys_key_str'~will~be~ignored.
9813      }
9814  \@@_msg_new:nnn { Unknown~key~for~custom-line }
9815      {
9816        Unknown~key.\\
9817        The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
9818        It~you~go~on,~you~will~probably~have~other~errors. \\
9819        \c_@@_available_keys_str
9820      }
9821      {
9822        The~available~keys~are~(in~alphabetic~order):~
9823        ccommand,~
9824        color,~
9825        command,~
9826        dotted,~
9827        letter,~
9828        multiplicity,~
9829        sep-color,~
9830        tikz,~and~total-width.
9831      }
9832  \@@_msg_new:nnn { Unknown~key~for~xdots }
9833      {
9834        Unknown~key.\\
9835        The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9836        \c_@@_available_keys_str
9837      }
9838      {
9839        The~available~keys~are~(in~alphabetic~order):~
9840        'color',~
9841        'horizontal-labels',~
9842        'inter',~
9843        'line-style',~
9844        'radius',~
9845        'shorten',~
9846        'shorten-end'~and~'shorten-start'.
9847      }
9848  \@@_msg_new:nn { Unknown~key~for~rowcolors }
9849      {
9850        Unknown~key.\\
9851        As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9852        (and~you~try~to~use~'\l_keys_key_str')\\
9853        That~key~will~be~ignored.
9854      }
9855  \@@_msg_new:nn { label~without~caption }
9856      {
```

```
9857        You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9858        you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9859      }
9860    \@@_msg_new:nn { W~warning }
9861      {
9862        Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9863        (row~\int_use:N \c@iRow).
9864      }
9865    \@@_msg_new:nn { Construct~too~large }
9866      {
9867        Construct~too~large.\\
9868        Your~command~\token_to_str:N #1
9869        can't~be~drawn~because~your~matrix~is~too~small.\\
9870        That~command~will~be~ignored.
9871      }
9872    \@@_msg_new:nn { underscore~after~nicematrix }
9873      {
9874        Problem~with~'underscore'.\\
9875        The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9876        You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9877        '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9878      }
9879    \@@_msg_new:nn { ampersand~in~light-syntax }
9880      {
9881        Ampersand~forbidden.\\
9882        You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9883        ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9884      }
9885    \@@_msg_new:nn { double-backslash~in~light-syntax }
9886      {
9887        Double~backslash~forbidden.\\
9888        You~can't~use~\token_to_str:N
9889        \\~to~separate~rows~because~the~key~'light-syntax'~
9890        is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9891        (set~by~the~key~'end-of-row').~This~error~is~fatal.
9892      }
9893    \@@_msg_new:nn { hlines~with~color }
9894      {
9895        Incompatible~keys.\\
9896        You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9897        '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9898        However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9899        Your~key~will~be~discarded.
9900      }
9901    \@@_msg_new:nn { bad~value~for~baseline }
9902      {
9903        Bad~value~for~baseline.\\
9904        The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9905        valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9906        \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9907        the~form~'line-i'.\\
9908        A~value~of~1~will~be~used.
9909      }
9910    \@@_msg_new:nn { detection~of~empty~cells }
9911      {
9912        Problem~with~'not-empty'\\
9913        For~technical~reasons,~you~must~activate~
9914        'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9915        in~order~to~use~the~key~'\l_keys_key_str'.\\
9916        That~key~will~be~ignored.
```

```
9917    }
9918  \@@_msg_new:nn { siunitx~not~loaded }
9919    {
9920      siunitx~not~loaded\\
9921      You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9922      That~error~is~fatal.
9923    }
9924  \@@_msg_new:nn { Invalid~name }
9925    {
9926      Invalid~name.\\
9927      You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9928      \SubMatrix\ of~your~\@@_full_name_env:.\\
9929      A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9930      This~key~will~be~ignored.
9931    }
9932  \@@_msg_new:nn { Hbrace~not~allowed }
9933    {
9934      Command~not~allowed.\\
9935      You~can't~use~the~command~\token_to_str:N \Hbrace\
9936      because~you~have~not~loaded~TikZ~
9937      and~the~TikZ~library~'decorations.pathreplacing'.\\
9938      Use:~\token_to_str:N \usepackage\{tikz\}~
9939      \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9940      That~command~will~be~ignored.
9941    }
9942  \@@_msg_new:nn { Vbrace~not~allowed }
9943    {
9944      Command~not~allowed.\\
9945      You~can't~use~the~command~\token_to_str:N \Vbrace\
9946      because~you~have~not~loaded~TikZ~
9947      and~the~TikZ~library~'decorations.pathreplacing'.\\
9948      Use:~\token_to_str:N \usepackage\{tikz\}~
9949      \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9950      That~command~will~be~ignored.
9951    }
9952  \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9953    {
9954      Wrong~line.\\
9955      You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9956      \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9957      number~is~not~valid.~It~will~be~ignored.
9958    }
9959  \@@_msg_new:nn { Impossible~delimiter }
9960    {
9961      Impossible~delimiter.\\
9962      It's~impossible~to~draw~the~#1~delimiter~of~your~
9963      \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9964      in~that~column.
9965      \bool_if:NT \l_@@_submatrix_slim_bool
9966        { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9967      This~\token_to_str:N \SubMatrix\ will~be~ignored.
9968    }
9969  \@@_msg_new:nnn { width~without~X~columns }
9970    {
9971      You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
9972     the~preamble~('\g_@@_user_preamble_tl')~of~your~\@@_full_name_env:.\\
9973      That~key~will~be~ignored.
9974    }
9975    {
9976      This~message~is~the~message~'width~without~X~columns'~
9977      of~the~module~'nicematrix'.~
```

```
9978        The~experimented~users~can~disable~that~message~with~
9979        \token_to_str:N \msg_redirect_name:nnn.\\
9980      }

9981

9982    \@@_msg_new:nn { key~multiplicity~with~dotted }
9983      {
9984        Incompatible~keys. \\
9985        You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9986        in~a~'custom-line'.~They~are~incompatible. \\
9987        The~key~'multiplicity'~will~be~discarded.
9988      }

9989    \@@_msg_new:nn { empty~environment }
9990      {
9991        Empty~environment.\\
9992        Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9993      }

9994    \@@_msg_new:nn { No~letter~and~no~command }
9995      {
9996        Erroneous~use.\\
9997        Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~
9998        key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9999        ~'ccommand'~(to~draw~horizontal~rules).\\
10000       However,~you~can~go~on.
10001     }

10002   \@@_msg_new:nn { Forbidden~letter }
10003     {
10004       Forbidden~letter.\\
10005       You~can't~use~the~letter~'#1'~for~a~customized~line.~
10006       It~will~be~ignored.\\
10007       The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10008     }

10009   \@@_msg_new:nn { Several~letters }
10010     {
10011       Wrong~name.\\
10012       You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10013       have~used~'\l_@@_letter_str').\\
10014       It~will~be~ignored.
10015     }

10016   \@@_msg_new:nn { Delimiter~with~small }
10017     {
10018       Delimiter~forbidden.\\
10019       You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
10020       because~the~key~'small'~is~in~force.\\
10021       This~error~is~fatal.
10022     }

10023   \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10024     {
10025       Unknown~cell.\\
10026       Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
10027       the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
10028       can't~be~executed~because~a~cell~doesn't~exist.\\
10029       This~command~\token_to_str:N \line\ will~be~ignored.
10030     }

10031   \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10032     {
10033       Duplicate~name.\\
10034       The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
10035       in~this~\@@_full_name_env:.\\
10036       This~key~will~be~ignored.\\
10037       \bool_if:NF \g_@@_messages_for_Overleaf_bool
```

```
10038          { For~a~list~of~the~names~already~used,~type~H~<return>. }
10039    }
10040    {
10041      The~names~already~defined~in~this~\@@_full_name_env:\ are:~
10042      \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
10043    }
10044  \@@_msg_new:nn { r~or~l~with~preamble }
10045    {
10046      Erroneous~use.\\
10047      You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:.~
10048      You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10049      your~\@@_full_name_env:.\\
10050      This~key~will~be~ignored.
10051    }
10052  \@@_msg_new:nn { Hdotsfor~in~col~0 }
10053    {
10054      Erroneous~use.\\
10055      You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
10056      the~array.~This~error~is~fatal.
10057    }
10058  \@@_msg_new:nn { bad~corner }
10059    {
10060      Bad~corner.\\
10061      #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
10062      'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
10063      This~specification~of~corner~will~be~ignored.
10064    }
10065  \@@_msg_new:nn { bad~border }
10066    {
10067      Bad~border.\\
10068      \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
10069      (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
10070      The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
10071      also~use~the~key~'tikz'
10072      \IfPackageLoadedF { tikz }
10073        {~if~you~load~the~LaTeX~package~'tikz'}).\\
10074      This~specification~of~border~will~be~ignored.
10075    }
10076  \@@_msg_new:nn { TikzEveryCell~without~tikz }
10077    {
10078      TikZ~not~loaded.\\
10079      You~can't~use~\token_to_str:N \TikzEveryCell\
10080      because~you~have~not~loaded~tikz.~
10081      This~command~will~be~ignored.
10082    }
10083  \@@_msg_new:nn { tikz~key~without~tikz }
10084    {
10085      TikZ~not~loaded.\\
10086      You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
10087      \Block'~because~you~have~not~loaded~tikz.~
10088      This~key~will~be~ignored.
10089    }
10090  \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
10091    {
10092      Erroneous~use.\\
10093      In~the~\@@_full_name_env:,~you~must~use~the~key~
10094      'last-col'~without~value.\\
10095      However,~you~can~go~on~for~this~time~
10096      (the~value~'\l_keys_value_tl'~will~be~ignored).
10097    }
```

```
10098  \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
10099    {
10100      Erroneous~use.\\
10101      In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
10102      'last-col'~without~value.\\
10103      However,~you~can~go~on~for~this~time~
10104      (the~value~'\l_keys_value_tl'~will~be~ignored).
10105    }
10106  \@@_msg_new:nn { Block~too~large~1 }
10107    {
10108      Block~too~large.\\
10109      You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
10110      too~small~for~that~block. \\
10111      This~block~and~maybe~others~will~be~ignored.
10112    }
10113  \@@_msg_new:nn { Block~too~large~2 }
10114    {
10115      Block~too~large.\\
10116      The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
10117      \g_@@_static_num_of_col_int\
10118      columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
10119      specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
10120      (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
10121      This~block~and~maybe~others~will~be~ignored.
10122    }
10123  \@@_msg_new:nn { unknown~column~type }
10124    {
10125      Bad~column~type.\\
10126      The~column~type~'#1'~in~your~\@@_full_name_env:\
10127      is~unknown. \\
10128      This~error~is~fatal.
10129    }
10130  \@@_msg_new:nn { unknown~column~type~S }
10131    {
10132      Bad~column~type.\\
10133      The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
10134      If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
10135      load~that~package. \\
10136      This~error~is~fatal.
10137    }
10138  \@@_msg_new:nn { tabularnote~forbidden }
10139    {
10140      Forbidden~command.\\
10141      You~can't~use~the~command~\token_to_str:N\tabularnote\
10142      ~here.~This~command~is~available~only~in~
10143      \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
10144      the~argument~of~a~command~\token_to_str:N \caption\ included~
10145      in~an~environment~{table}. \\
10146      This~command~will~be~ignored.
10147    }
10148  \@@_msg_new:nn { borders~forbidden }
10149    {
10150      Forbidden~key.\\
10151      You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
10152      because~the~option~'rounded-corners'~
10153      is~in~force~with~a~non-zero~value.\\
10154      This~key~will~be~ignored.
10155    }
10156  \@@_msg_new:nn { bottomrule~without~booktabs }
10157    {
10158      booktabs~not~loaded.\\
```

```
10159        You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
10160        loaded~'booktabs'.\\
10161        This~key~will~be~ignored.
10162      }
10163    \@@_msg_new:nn { enumitem~not~loaded }
10164      {
10165        enumitem~not~loaded.\\
10166        You~can't~use~the~command~\token_to_str:N\tabularnote\
10167        ~because~you~haven't~loaded~'enumitem'.\\
10168        All~the~commands~\token_to_str:N\tabularnote\ will~be~
10169        ignored~in~the~document.
10170      }
10171    \@@_msg_new:nn { tikz~without~tikz }
10172      {
10173        Tikz~not~loaded.\\
10174        You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
10175        loaded.~If~you~go~on,~that~key~will~be~ignored.
10176      }
10177    \@@_msg_new:nn { tikz~in~custom-line~without~tikz }
10178      {
10179        Tikz~not~loaded.\\
10180        You~have~used~the~key~'tikz'~in~the~definition~of~a~
10181        customized~line~(with~'custom-line')~but~tikz~is~not~loaded.~
10182        You~can~go~on~but~you~will~have~another~error~if~you~actually~
10183        use~that~custom-line.
10184      }
10185    \@@_msg_new:nn { tikz~in~borders~without~tikz }
10186      {
10187        Tikz~not~loaded.\\
10188        You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10189        command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
10190        That~key~will~be~ignored.
10191      }
10192    \@@_msg_new:nn { color~in~custom-line~with~tikz }
10193      {
10194        Erroneous~use.\\
10195        In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
10196        which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10197        The~key~'color'~will~be~discarded.
10198      }
10199    \@@_msg_new:nn { Wrong~last~row }
10200      {
10201        Wrong~number.\\
10202        You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
10203        \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
10204        If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
10205        last~row.~You~can~avoid~this~problem~by~using~'last-row'~
10206        without~value~(more~compilations~might~be~necessary).
10207      }
10208    \@@_msg_new:nn { Yet~in~env }
10209      {
10210        Nested~environments.\\
10211        Environments~of~nicematrix~can't~be~nested.\\
10212        This~error~is~fatal.
10213      }
10214    \@@_msg_new:nn { Outside~math~mode }
10215      {
10216        Outside~math~mode.\\
10217        The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
10218        (and~not~in~\token_to_str:N \vcenter).\\
```

```
10219      This~error~is~fatal.
10220    }
10221  \@@_msg_new:nn { One~letter~allowed }
10222    {
10223      Bad~name.\\
10224      The~value~of~key~'\l_keys_key_str'~must~be~of~length~1~and~
10225      you~have~used~'\l_keys_value_tl'.\\
10226      It~will~be~ignored.
10227    }
10228  \@@_msg_new:nn { TabularNote~in~CodeAfter }
10229    {
10230      Environment~{TabularNote}~forbidden.\\
10231      You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
10232      but~*before*~the~\token_to_str:N \CodeAfter.\\
10233      This~environment~{TabularNote}~will~be~ignored.
10234    }
10235  \@@_msg_new:nn { varwidth~not~loaded }
10236    {
10237      varwidth~not~loaded.\\
10238      You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10239      loaded.\\
10240      Your~column~will~behave~like~'p'.
10241    }
10242  \@@_msg_new:nnn { Unknow~key~for~RulesBis }
10243    {
10244      Unknown~key.\\
10245      Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
10246      \c_@@_available_keys_str
10247    }
10248    {
10249      The~available~keys~are~(in~alphabetic~order):~
10250      color,~
10251      dotted,~
10252      multiplicity,~
10253      sep-color,~
10254      tikz,~and~total-width.
10255    }
10256
10257  \@@_msg_new:nnn { Unknown~key~for~Block }
10258    {
10259      Unknown~key.\\
10260      The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10261      \Block.\\ It~will~be~ignored. \\
10262      \c_@@_available_keys_str
10263    }
10264    {
10265      The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
10266      b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10267      opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10268      and~vlines.
10269    }
10270  \@@_msg_new:nnn { Unknown~key~for~Brace }
10271    {
10272      Unknown~key.\\
10273      The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10274      \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10275      It~will~be~ignored. \\
10276      \c_@@_available_keys_str
10277    }
10278    {
10279      The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
```

```
10280      right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10281      right-shorten)~and~yshift.
10282    }
10283  \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10284    {
10285      Unknown~key.\\
10286      The~key~'\l_keys_key_str'~is~unknown.\\
10287      It~will~be~ignored. \\
10288      \c_@@_available_keys_str
10289    }
10290    {
10291      The~available~keys~are~(in~alphabetic~order):~
10292      delimiters/color,~
10293      rules~(with~the~subkeys~'color'~and~'width'),~
10294      sub-matrix~(several~subkeys)~
10295      and~xdots~(several~subkeys).~
10296      The~latter~is~for~the~command~\token_to_str:N \line.
10297    }
10298  \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10299    {
10300      Unknown~key.\\
10301      The~key~'\l_keys_key_str'~is~unknown.\\
10302      It~will~be~ignored. \\
10303      \c_@@_available_keys_str
10304    }
10305    {
10306      The~available~keys~are~(in~alphabetic~order):~
10307      create-cell-nodes,~
10308      delimiters/color~and~
10309      sub-matrix~(several~subkeys).
10310    }
10311  \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10312    {
10313      Unknown~key.\\
10314      The~key~'\l_keys_key_str'~is~unknown.\\
10315      That~key~will~be~ignored. \\
10316      \c_@@_available_keys_str
10317    }
10318    {
10319      The~available~keys~are~(in~alphabetic~order):~
10320      'delimiters/color',~
10321      'extra-height',~
10322      'hlines',~
10323      'hvlines',~
10324      'left-xshift',~
10325      'name',~
10326      'right-xshift',~
10327      'rules'~(with~the~subkeys~'color'~and~'width'),~
10328      'slim',~
10329      'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10330      and~'right-xshift').\\
10331    }
10332  \@@_msg_new:nnn { Unknown~key~for~notes }
10333    {
10334      Unknown~key.\\
10335      The~key~'\l_keys_key_str'~is~unknown.\\
10336      That~key~will~be~ignored. \\
10337      \c_@@_available_keys_str
10338    }
10339    {
10340      The~available~keys~are~(in~alphabetic~order):~
10341      bottomrule,~
```

```
10342      code-after,~
10343      code-before,~
10344      detect-duplicates,~
10345      enumitem-keys,~
10346      enumitem-keys-para,~
10347      para,~
10348      label-in-list,~
10349      label-in-tabular~and~
10350      style.
10351    }
10352  \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10353    {
10354      Unknown~key.\\
10355      The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10356      \token_to_str:N \RowStyle. \\
10357      That~key~will~be~ignored. \\
10358      \c_@@_available_keys_str
10359    }
10360    {
10361      The~available~keys~are~(in~alphabetic~order):~
10362      bold,~
10363      cell-space-top-limit,~
10364      cell-space-bottom-limit,~
10365      cell-space-limits,~
10366      color,~
10367      fill~(alias:~rowcolor),~
10368      nb-rows,
10369      opacity~and~
10370      rounded-corners.
10371    }
10372  \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10373    {
10374      Unknown~key.\\
10375      The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10376      \token_to_str:N \NiceMatrixOptions. \\
10377      That~key~will~be~ignored. \\
10378      \c_@@_available_keys_str
10379    }
10380    {
10381      The~available~keys~are~(in~alphabetic~order):~
10382      &-in-blocks,~
10383      allow-duplicate-names,~
10384      ampersand-in-blocks,~
10385      caption-above,~
10386      cell-space-bottom-limit,~
10387      cell-space-limits,~
10388      cell-space-top-limit,~
10389      code-for-first-col,~
10390      code-for-first-row,~
10391      code-for-last-col,~
10392      code-for-last-row,~
10393      corners,~
10394      custom-key,~
10395      create-extra-nodes,~
10396      create-medium-nodes,~
10397      create-large-nodes,~
10398      custom-line,~
10399      delimiters~(several~subkeys),~
10400      end-of-row,~
10401      first-col,~
10402      first-row,~
10403      hlines,~
10404      hvlines,~
```

```
10405      hvlines-except-borders,~
10406      last-col,~
10407      last-row,~
10408      left-margin,~
10409      light-syntax,~
10410      light-syntax-expanded,~
10411      matrix/columns-type,~
10412      no-cell-nodes,~
10413      notes~(several~subkeys),~
10414      nullify-dots,~
10415      pgf-node-code,~
10416      renew-dots,~
10417      renew-matrix,~
10418      respect-arraystretch,~
10419      rounded-corners,~
10420      right-margin,~
10421      rules~(with~the~subkeys~'color'~and~'width'),~
10422      small,~
10423      sub-matrix~(several~subkeys),~
10424      vlines,~
10425      xdots~(several~subkeys).
10426    }
```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and
r.

```
10427  \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10428    {
10429      Unknown~key.\\
10430      The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10431      \{NiceArray\}. \\
10432      That~key~will~be~ignored. \\
10433      \c_@@_available_keys_str
10434    }
10435    {
10436      The~available~keys~are~(in~alphabetic~order):~
10437      &-in-blocks,~
10438      ampersand-in-blocks,~
10439      b,~
10440      baseline,~
10441      c,~
10442      cell-space-bottom-limit,~
10443      cell-space-limits,~
10444      cell-space-top-limit,~
10445      code-after,~
10446      code-for-first-col,~
10447      code-for-first-row,~
10448      code-for-last-col,~
10449      code-for-last-row,~
10450      columns-width,~
10451      corners,~
10452      create-extra-nodes,~
10453      create-medium-nodes,~
10454      create-large-nodes,~
10455      extra-left-margin,~
10456      extra-right-margin,~
10457      first-col,~
10458      first-row,~
10459      hlines,~
10460      hvlines,~
10461      hvlines-except-borders,~
10462      last-col,~
10463      last-row,~
10464      left-margin,~
10465      light-syntax,~
```

```
10466      light-syntax-expanded,~
10467      name,~
10468      no-cell-nodes,~
10469      nullify-dots,~
10470      pgf-node-code,~
10471      renew-dots,~
10472      respect-arraystretch,~
10473      right-margin,~
10474      rounded-corners,~
10475      rules~(with~the~subkeys~'color'~and~'width'),~
10476      small,~
10477      t,~
10478      vlines,~
10479      xdots/color,~
10480      xdots/shorten-start,~
10481      xdots/shorten-end,~
10482      xdots/shorten~and~
10483      xdots/line-style.
10484    }
```

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`).

```
10485  \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10486    {
10487      Unknown~key.\\
10488      The~key~'\l_keys_key_str'~is~unknown~for~the~
10489      \@@_full_name_env:. \\
10490      That~key~will~be~ignored. \\
10491      \c_@@_available_keys_str
10492    }
10493    {
10494      The~available~keys~are~(in~alphabetic~order):~
10495      &-in-blocks,~
10496      ampersand-in-blocks,~
10497      b,~
10498      baseline,~
10499      c,~
10500      cell-space-bottom-limit,~
10501      cell-space-limits,~
10502      cell-space-top-limit,~
10503      code-after,~
10504      code-for-first-col,~
10505      code-for-first-row,~
10506      code-for-last-col,~
10507      code-for-last-row,~
10508      columns-type,~
10509      columns-width,~
10510      corners,~
10511      create-extra-nodes,~
10512      create-medium-nodes,~
10513      create-large-nodes,~
10514      extra-left-margin,~
10515      extra-right-margin,~
10516      first-col,~
10517      first-row,~
10518      hlines,~
10519      hvlines,~
10520      hvlines-except-borders,~
10521      l,~
10522      last-col,~
10523      last-row,~
10524      left-margin,~
10525      light-syntax,~
10526      light-syntax-expanded,~
```

```
10527        name,~
10528        no-cell-nodes,~
10529        nullify-dots,~
10530        pgf-node-code,~
10531        r,~
10532        renew-dots,~
10533        respect-arraystretch,~
10534        right-margin,~
10535        rounded-corners,~
10536        rules~(with~the~subkeys~'color'~and~'width'),~
10537        small,~
10538        t,~
10539        vlines,~
10540        xdots/color,~
10541        xdots/shorten-start,~
10542        xdots/shorten-end,~
10543        xdots/shorten~and~
10544        xdots/line-style.
10545      }
10546    \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10547      {
10548        Unknown~key.\\
10549        The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10550        \{NiceTabular\}. \\
10551        That~key~will~be~ignored. \\
10552        \c_@@_available_keys_str
10553      }
10554      {
10555        The~available~keys~are~(in~alphabetic~order):~
10556        &-in-blocks,~
10557        ampersand-in-blocks,~
10558        b,~
10559        baseline,~
10560        c,~
10561        caption,~
10562        cell-space-bottom-limit,~
10563        cell-space-limits,~
10564        cell-space-top-limit,~
10565        code-after,~
10566        code-for-first-col,~
10567        code-for-first-row,~
10568        code-for-last-col,~
10569        code-for-last-row,~
10570        columns-width,~
10571        corners,~
10572        custom-line,~
10573        create-extra-nodes,~
10574        create-medium-nodes,~
10575        create-large-nodes,~
10576        extra-left-margin,~
10577        extra-right-margin,~
10578        first-col,~
10579        first-row,~
10580        hlines,~
10581        hvlines,~
10582        hvlines-except-borders,~
10583        label,~
10584        last-col,~
10585        last-row,~
10586        left-margin,~
10587        light-syntax,~
10588        light-syntax-expanded,~
10589        name,~
```

```
10590      no-cell-nodes,~
10591      notes~(several~subkeys),~
10592      nullify-dots,~
10593      pgf-node-code,~
10594      renew-dots,~
10595      respect-arraystretch,~
10596      right-margin,~
10597      rounded-corners,~
10598      rules~(with~the~subkeys~'color'~and~'width'),~
10599      short-caption,~
10600      t,~
10601      tabularnote,~
10602      vlines,~
10603      xdots/color,~
10604      xdots/shorten-start,~
10605      xdots/shorten-end,~
10606      xdots/shorten~and~
10607      xdots/line-style.
10608    }
10609  \@@_msg_new:nnn { Duplicate~name }
10610    {
10611      Duplicate~name.\\
10612      The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10613      the~same~environment~name~twice.~You~can~go~on,~but,~
10614      maybe,~you~will~have~incorrect~results~especially~
10615      if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10616      message~again,~use~the~key~'allow-duplicate-names'~in~
10617      '\token_to_str:N \NiceMatrixOptions'.\\
10618      \bool_if:NF \g_@@_messages_for_Overleaf_bool
10619        { For~a~list~of~the~names~already~used,~type~H~<return>. }
10620    }
10621    {
10622      The~names~already~defined~in~this~document~are:~
10623      \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10624    }
10625  \@@_msg_new:nn { Option~auto~for~columns-width }
10626    {
10627      Erroneous~use.\\
10628      You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10629      That~key~will~be~ignored.
10630    }
10631  \@@_msg_new:nn { NiceTabularX~without~X }
10632    {
10633      NiceTabularX~without~X.\\
10634      You~should~not~use~{NiceTabularX}~without~X~columns.\\
10635      However,~you~can~go~on.
10636    }
10637  \@@_msg_new:nn { Preamble~forgotten }
10638    {
10639      Preamble~forgotten.\\
10640      You~have~probably~forgotten~the~preamble~of~your~
10641      \@@_full_name_env:. \\
10642      This~error~is~fatal.
10643    }
10644  \@@_msg_new:nn { Invalid~col~number }
10645    {
10646      Invalid~column~number.\\
10647      A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10648      specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10649    }
10650  \@@_msg_new:nn { Invalid~row~number }
```

```
10651    {
10652      Invalid~row~number.\\
10653      A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10654      specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10655    }
```

Contents