

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

March 30, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

*This document corresponds to the version 7.1b of `nicematrix`, at the date of 2025/03/30.

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

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

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

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

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

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

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

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

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

```
43 \cs_new_protected:Npn \@@_error_or_warning:n
44 { \bool_if:NTF \g_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }
```

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

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

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

```

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

```

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

```

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

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

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

```

2 Collecting options

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

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`).

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

106 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

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

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

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

Idem for `\CT@drs@`.

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

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

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

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

```
217     \skip_horizontal:N \c_zero_dim
218   }
```

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

223 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

287 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
302 \dim_new:N \l_@@_col_width_dim
303 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
304 \int_new:N \g_@@_row_total_int
305 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
308 \tl_new:N \l_@@_hpos_cell_tl
309 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
311 \dim_new:N \g_@@_blocks_ht_dim
```

```
312 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

```
317 \bool_set_true:N \l_@@_notes_detect_duplicates_bool
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
330 \tl_new:N \g_@@_right_delim_tl
```

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

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

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

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

For `\multicolumn`.

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

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

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

```
335 \str_set:Nn \l_@@_columns_type_tl { c }
```

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

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

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

```
338 \tl_new:N \l_@@_xdots_middle_tl
```

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

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

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

```
341 {
```

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

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

```
344   \fi:
```

```
345 }
```

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

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

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

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

```
348 \colorlet { nicematrix-last-row } { . }
```

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

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

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

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

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

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

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

```
359 \tl_new:N \g_@@_cell_after_hook_tl % 2025/03/22
```

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

```
360 \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.

```
361 \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.

```
362 \tl_new:N \g_@@_pre_code_before_tl
363 \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.

```
364 \tl_new:N \g_@@_pre_code_after_tl
365 \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.

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

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

```
367 \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.

```
368 \int_new:N \l_@@_old_iRow_int
369 \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).

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

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

```
371 \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.

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

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

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

```
374 \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.

```
375 \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).

```
376 \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).

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

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

```
379 \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).

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

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

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

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

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

```
384 \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.

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

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

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

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

```

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

```

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

```

395 \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”.

```

396 \dim_new:N \g_@@_width_last_col_dim
397 \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.

```

398 \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!).

```

399 \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.

```

400 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

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

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

```

401 \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.

```

402 \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`).

```

403 \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`).

```

404 \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.

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

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

```
406 \seq_new:N \g_@@_multicolumn_cells_seq
407 \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).

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

The following counters will be used when drawing the rules.

```
412 \int_new:N \l_@@_start_int
413 \int_set_eq:NN \l_@@_start_int \c_one_int
414 \int_new:N \l_@@_end_int
415 \int_new:N \l_@@_local_start_int
416 \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.

```
417 \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).

```
418 \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`.

```
419 \tl_new:N \l_@@_fill_tl
420 \tl_new:N \l_@@_opacity_tl
421 \tl_new:N \l_@@_draw_tl
422 \seq_new:N \l_@@_tikz_seq
423 \clist_new:N \l_@@_borders_clist
424 \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.

```
425 \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`.

```
426 \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.

```
427 \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.

```
428 \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`).

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

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

```
433 \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).

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

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

```
435 \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).

```
436 \bool_new:N \l_@@_vlines_block_bool
437 \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.

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

439 \dim_new:N \l_@@_submatrix_extra_height_dim
440 \dim_new:N \l_@@_submatrix_left_xshift_dim
441 \dim_new:N \l_@@_submatrix_right_xshift_dim
442 \clist_new:N \l_@@_hlines_clist
443 \clist_new:N \l_@@_vlines_clist
444 \clist_new:N \l_@@_submatrix_hlines_clist
445 \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}`).

```
446 \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.

```
447 \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`).

```
448 \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.

```
449 \int_new:N \l_@@_first_row_int
450 \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.

```
451 \int_new:N \l_@@_first_col_int
452 \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).

```
453 \int_new:N \l_@@_last_row_int
454 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
456 \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.

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

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

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

```

\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.

```
459 \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`).

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

Some utilities

```
461 \cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
462 {
```

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

```
463 \cs_set_nopar:Npn \l_tmpa_tl { #1 }
464 \cs_set_nopar:Npn \l_tmpb_tl { #2 }
465 }
```

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.

```
466 \cs_new_protected:Npn \@@_expand_clist:N #1
467 {
468 \clist_if_in:NnF #1 { all }
469 {
470 \clist_clear:N \l_tmpa_clist
471 \clist_map_inline:Nn #1
472 {
```

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

```
473 \tl_if_in:nnTF { ##1 } { - }
474 { \@@_cut_on_hyphen:w ##1 \q_stop }
475 {
```

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

```
476 \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
477 \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
478 }
479 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
480 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
481 }
482 \tl_set_eq:NN #1 \l_tmpa_clist
483 }
484 }
```

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.

```
485 \hook_gput_code:nnn { begindocument } { . }
486 {
487 \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
488 \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
489 \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
490 }
```

5 The command `\tabularnote`

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

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

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

```
491 \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`.

```
492 \int_new:N \g_@@_tabularnote_int
```

```
493 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
```

```
494 \seq_new:N \g_@@_notes_seq
```

```
495 \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.

```
496 \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.

```
497 \seq_new:N \l_@@_notes_labels_seq
```

```
498 \newcounter { nicematrix_draft }
```

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

```

499 \cs_new_protected:Npn \@@_notes_format:n #1
500 {
501   \setcounter { nicematrix_draft } { #1 }
502   \@@_notes_style:n { nicematrix_draft }
503 }

```

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

```

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

```

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

```

505 \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`.

```

506 \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`.

```

507 \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`).

```

508 \hook_gput_code:nnn { begindocument } { . }
509 {
510   \IfPackageLoadedTF { enumitem }
511   {

```

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.

```

512     \newlist { tabularnotes } { enumerate } { 1 }
513     \setlist [ tabularnotes ]
514     {
515       topsep = 0pt ,
516       noitemsep ,
517       leftmargin = * ,
518       align = left ,
519       labelsep = 0pt ,
520       label =
521         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
522     }
523     \newlist { tabularnotes* } { enumerate* } { 1 }
524     \setlist* [ tabularnotes* ]
525     {
526       afterlabel = \nobreak ,
527       itemjoin = \quad ,
528       label =
529         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
530     }

```

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`.

```

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

```

```

534     {
535         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
536         { \@@_error:n { tabularnote~forbidden } }
537     {
538         \bool_if:NTF \l_@@_in_caption_bool
539         \@@_tabularnote_caption:nn
540         \@@_tabularnote:nn
541         { #1 } { #2 }
542     }
543 }
544 }
545 }
546 {
547     \NewDocumentCommand \tabularnote { o m }
548     {
549         \@@_error_or_warning:n { enumitem~not~loaded }
550         \@@_gredirect_none:n { enumitem~not~loaded }
551     }
552 }
553 }
554 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
555 { \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`.

```

556 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
557 {

```

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).

```

558     \int_zero:N \l_tmpa_int
559     \bool_if:NT \l_@@_notes_detect_duplicates_bool
560     {

```

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`.

```

561     \int_zero:N \l_tmpb_int
562     \seq_map_indexed_inline:Nn \g_@@_notes_seq
563     {
564         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
565         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
566         {
567             \tl_if_novalue:nTF { #1 }
568             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
569             { \int_set:Nn \l_tmpa_int { ##1 } }
570         }
571     }
572 }
573 \int_if_zero:nF \l_tmpa_int
574 { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
575 }
576 \int_if_zero:nT \l_tmpa_int
577 {

```

```

578     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
579     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
580   }
581   \seq_put_right:Ne \l_@@_notes_labels_seq
582   {
583     \tl_if_novalue:nTF { #1 }
584     {
585       \@@_notes_format:n
586       {
587         \int_eval:n
588         {
589           \int_if_zero:nTF \l_tmpa_int
590           \c@tabularnote
591           \l_tmpa_int
592         }
593       }
594     }
595     { #1 }
596   }
597   \peek_meaning:NF \tabularnote
598   {

```

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`.

```

599     \hbox_set:Nn \l_tmpa_box
600     {

```

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

```

601     \@@_notes_label_in_tabular:n
602     {
603       \seq_use:Nnnn
604       \l_@@_notes_labels_seq { , } { , } { , }
605     }
606   }

```

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

```

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

```

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

```

609     \int_gincr:N \g_@@_tabularnote_int
610     \refstepcounter { tabularnote }
611     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
612     { \int_gincr:N \c@tabularnote }
613     \seq_clear:N \l_@@_notes_labels_seq
614     \bool_lazy_or:nnTF
615     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
616     { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
617     {
618       \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).

```

619     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
620   }
621   { \box_use:N \l_tmpa_box }
622 }
623 }

```

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`.

```

624 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
625   {
626     \bool_if:NTF \g_@@_caption_finished_bool
627       {
628         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
629         { \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`!

```

630     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
631     { \@@_error:n { Identical-notes-in-caption } }
632   }
633   {

```

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

```

634     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
635     {

```

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.

```

636         \bool_gset_true:N \g_@@_caption_finished_bool
637         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
638         \int_gzero:N \c@tabularnote
639       }
640     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
641   }

```

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!

```

642   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
643   \seq_put_right:Ne \l_@@_notes_labels_seq
644   {
645     \tl_if_novalue:nTF { #1 }
646     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
647     { #1 }
648   }
649   \peek_meaning:NF \tabularnote
650   {
651     \@@_notes_label_in_tabular:n
652     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
653     \seq_clear:N \l_@@_notes_labels_seq
654   }
655 }

656 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
657 { \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.

```

658 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
659 {
660   \begin { pgfscope }
661   \pgfset
662   {
663     inner~sep = \c_zero_dim ,
664     minimum~size = \c_zero_dim
665   }
666   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
667   \pgfnode
668   { rectangle }
669   { center }
670   {
671     \vbox_to_ht:nn
672     { \dim_abs:n { #5 - #3 } }
673     {
674       \vfill
675       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
676     }
677   }
678   { #1 }
679   { }
680   \end { pgfscope }
681 }

```

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.

```

682 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
683 {
684   \begin { pgfscope }
685   \pgfset
686   {
687     inner~sep = \c_zero_dim ,
688     minimum~size = \c_zero_dim
689   }
690   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
691   \pgfpointdiff { #3 } { #2 }
692   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
693   \pgfnode
694   { rectangle }
695   { center }
696   {
697     \vbox_to_ht:nn
698     { \dim_abs:n \l_tmpb_dim }
699     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
700   }
701   { #1 }
702   { }
703   \end { pgfscope }
704 }

```

7 The options

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

```

705 \tl_new:N \l_@@_caption_tl
706 \tl_new:N \l_@@_short_caption_tl
707 \tl_new:N \l_@@_label_tl

```


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

```
708 \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`.

```
709 \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`).

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

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

```
712 \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.

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

```
716 \dim_new:N \l_@@_xdots_shorten_start_dim
717 \dim_new:N \l_@@_xdots_shorten_end_dim
718 \hook_gput_code:nnn { begindocument } { . }
719 {
720   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
721   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
722 }
```

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.

```
723 \dim_new:N \l_@@_xdots_radius_dim
724 \hook_gput_code:nnn { begindocument } { . }
725 { \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.

```
726 \tl_new:N \l_@@_xdots_line_style_tl
727 \tl_const:Nn \c_@@_standard_tl { standard }
728 \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`.

```
729 \bool_new:N \l_@@_light_syntax_bool
730 \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).

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

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

```
733 \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`).

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

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

```
735 \bool_new:N \l_@@_parallelize_diags_bool
736 \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`.

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

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

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

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

```
742 \cs_new_protected:Npn \@@_reset_arraystretch:
743 { \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).

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

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

```
745 \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.

```
746 \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”.

```
747 \bool_new:N \l_@@_medium_nodes_bool
748 \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).

```
749 \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`).

```
750 \dim_new:N \l_@@_left_margin_dim
751 \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`.

```
752 \dim_new:N \l_@@_extra_left_margin_dim
753 \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.

```
754 \tl_new:N \l_@@_end_of_row_tl
755 \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 “:”.

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

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

```
757 \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.

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

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

```

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

```

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

```

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

```

The key `draw-first`, which is meant to be used only with `\Ddots` and `\Iddots`, will be caught when `\Ddots` or `\Iddots` is used (during the construction of the array and not when we draw the dotted lines).

```

800   draw-first .code:n = \prg_do_nothing: ,
801   unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
802 }

```

```

803 \keys_define:nn { nicematrix / rules }
804 {
805   color .tl_set:N = \l_@@_rules_color_tl ,
806   color .value_required:n = true ,
807   width .dim_set:N = \arrayrulewidth ,
808   width .value_required:n = true ,
809   unknown .code:n = \@@_error:n { Unknown-key-for-rules }
810 }

```

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

```

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

```

```

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

```

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

```

875   { \cs_set_eq:cN { @@_#1 } \@@_make_preamble_vlism:n }
876   }
877   { \@@_error:n { One~letter~allowed } }
878   } ,
879 vlines-in-sub-matrix .value_required:n = true ,
880 hvlines .code:n =
881 {
882   \bool_set_true:N \l_@@_hvlines_bool
883   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
884   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
885   } ,
886 hvlines-except-borders .code:n =
887 {
888   \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
889   \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
890   \bool_set_true:N \l_@@_hvlines_bool
891   \bool_set_true:N \l_@@_except_borders_bool
892   } ,
893 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.

```

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

```

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

```

918 \keys_define:nm { nicematrix / environments }
919 {
920   corners .clist_set:N = \l_@@_corners_clist ,
921   corners .default:n = { NW , SW , NE , SE } ,
922   code-before .code:n =
923     {
924       \tl_if_empty:nF { #1 }
925       {
926         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
927         \bool_set_true:N \l_@@_code_before_bool
928       }
929     } ,
930   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}`.

```

931   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
932   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
933   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
934   baseline .tl_set:N = \l_@@_baseline_tl ,
935   baseline .value_required:n = true ,
936   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)`.

```

937   \str_if_eq:eeTF { #1 } { auto }
938   { \bool_set_true:N \l_@@_auto_columns_width_bool }
939   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
940   columns-width .value_required:n = true ,
941   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.

```

942 \legacy_if:nF { measuring@ }
943 {
944   \str_set:Ne \l_tmpa_str { #1 }
945   \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
946   { \@@_error:nn { Duplicate~name } { #1 } }
947   { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
948   \str_set_eq:NN \l_@@_name_str \l_tmpa_str
949   } ,
950   name .value_required:n = true ,
951   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
952   code-after .value_required:n = true ,
953 }
954 \keys_define:nn { nicematrix / notes }
955 {
956   para .bool_set:N = \l_@@_notes_para_bool ,
957   para .default:n = true ,
958   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
959   code-before .value_required:n = true ,
960   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
961   code-after .value_required:n = true ,
962   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
963   bottomrule .default:n = true ,
964   style .cs_set:Np = \@@_notes_style:n #1 ,
965   style .value_required:n = true ,
966   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
967   label-in-tabular .value_required:n = true ,
968   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
969   label-in-list .value_required:n = true ,
970   enumitem-keys .code:n =
971   {
972     \hook_gput_code:nnn { begindocument } { . }
973     {
974       \IfPackageLoadedT { enumitem }
975       { \setlist* [ tabularnotes ] { #1 } }
976     }
977   } ,
978   enumitem-keys .value_required:n = true ,
979   enumitem-keys-para .code:n =
980   {
981     \hook_gput_code:nnn { begindocument } { . }
982     {
983       \IfPackageLoadedT { enumitem }
984       { \setlist* [ tabularnotes* ] { #1 } }
985     }
986   } ,
987   enumitem-keys-para .value_required:n = true ,
988   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
989   detect-duplicates .default:n = true ,
990   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
991 }
992 \keys_define:nn { nicematrix / delimiters }
993 {
994   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
995   max-width .default:n = true ,
996   color .tl_set:N = \l_@@_delimiters_color_tl ,
997   color .value_required:n = true ,
998 }

```

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

```

999 \keys_define:nn { nicematrix }
1000 {

```

```

1001 NiceMatrixOptions .inherit:n =
1002   { nicematrix / Global } ,
1003 NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1004 NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1005 NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1006 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1007 SubMatrix / rules .inherit:n = nicematrix / rules ,
1008 CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1009 CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1010 CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1011 NiceMatrix .inherit:n =
1012   {
1013     nicematrix / Global ,
1014     nicematrix / environments ,
1015   } ,
1016 NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1017 NiceMatrix / rules .inherit:n = nicematrix / rules ,
1018 NiceTabular .inherit:n =
1019   {
1020     nicematrix / Global ,
1021     nicematrix / environments
1022   } ,
1023 NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1024 NiceTabular / rules .inherit:n = nicematrix / rules ,
1025 NiceTabular / notes .inherit:n = nicematrix / notes ,
1026 NiceArray .inherit:n =
1027   {
1028     nicematrix / Global ,
1029     nicematrix / environments ,
1030   } ,
1031 NiceArray / xdots .inherit:n = nicematrix / xdots ,
1032 NiceArray / rules .inherit:n = nicematrix / rules ,
1033 pNiceArray .inherit:n =
1034   {
1035     nicematrix / Global ,
1036     nicematrix / environments ,
1037   } ,
1038 pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1039 pNiceArray / rules .inherit:n = nicematrix / rules ,
1040 }

```

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

```

1041 \keys_define:nm { nicematrix / NiceMatrixOptions }
1042 {
1043   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1044   delimiters / color .value_required:n = true ,
1045   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1046   delimiters / max-width .default:n = true ,
1047   delimiters .code:n = \keys_set:nm { nicematrix / delimiters } { #1 } ,
1048   delimiters .value_required:n = true ,
1049   width .dim_set:N = \l_@@_width_dim ,
1050   width .value_required:n = true ,
1051   last-col .code:n =
1052     \tl_if_empty:nF { #1 }
1053     { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1054     \int_zero:N \l_@@_last_col_int ,
1055   small .bool_set:N = \l_@@_small_bool ,
1056   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.

```

1057   renew-matrix .code:n = \@@_renew_matrix: ,
1058   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}`.

```
1059 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.

```
1060 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)`.

```
1061 \str_if_eq:eeTF { #1 } { auto }
1062 { \@@_error:n { Option~auto~for~columns~width } }
1063 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

```
1064 allow-duplicate-names .code:n =
1065 \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1066 allow-duplicate-names .value_forbidden:n = true ,
1067 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1068 notes .value_required:n = true ,
1069 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1070 sub-matrix .value_required:n = true ,
1071 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1072 matrix / columns-type .value_required:n = true ,
1073 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1074 caption-above .default:n = true ,
1075 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1076 }
```

`\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.

```
1077 \NewDocumentCommand \NiceMatrixOptions { m }
1078 { \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.

```
1079 \keys_define:nn { nicematrix / NiceMatrix }
1080 {
1081 last-col .code:n = \tl_if_empty:nTF { #1 }
1082 {
1083 \bool_set_true:N \l_@@_last_col_without_value_bool
1084 \int_set:Nn \l_@@_last_col_int { -1 }
1085 }
1086 { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1087 columns-type .tl_set:N = \l_@@_columns_type_tl ,
1088 columns-type .value_required:n = true ,
1089 l .meta:n = { columns-type = l } ,
1090 r .meta:n = { columns-type = r } ,
1091 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1092 delimiters / color .value_required:n = true ,
1093 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1094 delimiters / max-width .default:n = true ,
1095 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1096 delimiters .value_required:n = true ,
1097 small .bool_set:N = \l_@@_small_bool ,
1098 small .value_forbidden:n = true ,
1099 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1100 }
```

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

```
1101 \keys_define:nn { nicematrix / NiceArray }
1102 {
```

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.

```
1103     small .bool_set:N = \l_@@_small_bool ,
1104     small .value_forbidden:n = true ,
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     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1109     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1110     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1111 }
```

```
1112 \keys_define:nn { nicematrix / pNiceArray }
1113 {
1114     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1115     last-col .code:n = \tl_if_empty:nF { #1 }
1116                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1117                 \int_zero:N \l_@@_last_col_int ,
1118     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1119     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1120     delimiters / color .value_required:n = true ,
1121     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1122     delimiters / max-width .default:n = true ,
1123     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1124     delimiters .value_required:n = true ,
1125     small .bool_set:N = \l_@@_small_bool ,
1126     small .value_forbidden:n = true ,
1127     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1128     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1129     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1130 }
```

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

```
1131 \keys_define:nn { nicematrix / NiceTabular }
1132 {
```

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.

```
1133     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1134                 \bool_set_true:N \l_@@_width_used_bool ,
1135     width .value_required:n = true ,
1136     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1137     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1138     tabularnote .value_required:n = true ,
1139     caption .tl_set:N = \l_@@_caption_tl ,
1140     caption .value_required:n = true ,
1141     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1142     short-caption .value_required:n = true ,
1143     label .tl_set:N = \l_@@_label_tl ,
1144     label .value_required:n = true ,
1145     last-col .code:n = \tl_if_empty:nF { #1 }
1146                 { \@@_error:n { last-col-non-empty-for-NiceArray } }
1147                 \int_zero:N \l_@@_last_col_int ,
1148     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1149     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1150     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1151 }
```

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

1152 \keys_define:nn { nicematrix / CodeAfter }
1153 {
1154   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1155   delimiters / color .value_required:n = true ,
1156   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1157   rules .value_required:n = true ,
1158   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1159   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1160   sub-matrix .value_required:n = true ,
1161   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1162 }
```

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}`).

```
1163 \cs_new_protected:Npn \@@_cell_begin:
1164 {
```

`\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.

```
1165   \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).

```
1166   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1167   \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.

```
1168   \int_compare:nNnT \c@jCol = \c_one_int
1169     { \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:`.

```
1170   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1171   \@@_tuning_not_tabular_begin:
1172   \@@_tuning_first_row:
1173   \@@_tuning_last_row:
1174   \g_@@_row_style_tl
1175 }
```

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.

```

1176 \cs_new_protected:Npn \@@_tuning_first_row:
1177 {
1178   \if_int_compare:w \c@iRow = \c_zero_int
1179     \if_int_compare:w \c@jCol > \c_zero_int
1180       \l_@@_code_for_first_row_tl
1181       \xglobal \colorlet { nicematrix-first-row } { . }
1182     \fi:
1183   \fi:
1184 }

```

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.

```

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

```

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

```

1192 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1193 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1194 {
1195   \m@th % added 2024/11/21
1196   \c_math_toggle_token

```

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

```

1197   \@@_tuning_key_small:
1198 }
1199 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```

1200 \cs_new_protected:Npn \@@_begin_of_row:
1201 {
1202   \int_gincr:N \c@iRow

```

```

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

```

Remark: If the key `recreate-cell-nodes` of the `\CodeBefore` is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the `last-row`, some lines of code will be dynamically added to this command.

```

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

```

```

1259     }
1260     \bool_gset_false:N \g_@@_rotate_bool
1261     \bool_gset_false:N \g_@@_rotate_c_bool
1262   }
1263 \cs_new_protected:Npn \@@_adjust_size_box:
1264 {
1265   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1266   {
1267     \box_set_wd:Nn \l_@@_cell_box
1268     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1269     \dim_gzero:N \g_@@_blocks_wd_dim
1270   }
1271   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1272   {
1273     \box_set_dp:Nn \l_@@_cell_box
1274     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1275     \dim_gzero:N \g_@@_blocks_dp_dim
1276   }
1277   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1278   {
1279     \box_set_ht:Nn \l_@@_cell_box
1280     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1281     \dim_gzero:N \g_@@_blocks_ht_dim
1282   }
1283 }
1284 \cs_new_protected:Npn \@@_cell_end:
1285 {

```

The following command is nullified in the tabulars.

```

1286   \@@_tuning_not_tabular_end:
1287   \hbox_set_end:
1288   \@@_cell_end_i:
1289 }
1290 \cs_new_protected:Npn \@@_cell_end_i:
1291 {

```

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.

```

1292   \g_@@_cell_after_hook_tl
1293   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1294   \@@_adjust_size_box:
1295   \box_set_ht:Nn \l_@@_cell_box
1296   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1297   \box_set_dp:Nn \l_@@_cell_box
1298   { \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”).

```

1299   \@@_update_max_cell_width:

```

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

```

1300   \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

1301   \bool_if:NTF \g_@@_empty_cell_bool
1302     { \box_use_drop:N \l_@@_cell_box }
1303     {
1304       \bool_if:NTF \g_@@_not_empty_cell_bool
1305         \@@_print_node_cell:
1306         {
1307           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1308             \@@_print_node_cell:
1309             { \box_use_drop:N \l_@@_cell_box }
1310         }
1311     }
1312   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1313     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1314   \bool_gset_false:N \g_@@_empty_cell_bool
1315   \bool_gset_false:N \g_@@_not_empty_cell_bool
1316 }

```

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

```

1317 \cs_new_protected:Npn \@@_update_max_cell_width:
1318   {
1319     \dim_gset:Nn \g_@@_max_cell_width_dim
1320     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1321   }

```

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

```

1322 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1323   {
1324     \@@_math_toggle:
1325     \hbox_set_end:
1326     \bool_if:NF \g_@@_rotate_bool
1327       {
1328         \hbox_set:Nn \l_@@_cell_box
1329           {
1330             \makebox [ \l_@@_col_width_dim ] [ s ]
1331             { \hbox_unpack_drop:N \l_@@_cell_box }
1332           }
1333       }
1334     \@@_cell_end_i:
1335   }

```

```

1336 \pgfset
1337   {
1338     nicematrix / cell-node /.style =
1339     {
1340       inner~sep = \c_zero_dim ,
1341       minimum~width = \c_zero_dim
1342     }
1343   }

```

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

```

1344 \socket_new:nn { nicematrix / siunitx-wrap } { 1 }
1345 \socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
1346 {
1347   \use:c
1348   {
1349     __siunitx_table_align_
1350     \bool_if:NTF \l__siunitx_table_text_bool
1351     \l__siunitx_table_align_text_tl
1352     \l__siunitx_table_align_number_tl
1353     :n
1354   }
1355   { #1 }
1356 }
1357 \cs_new_protected:Npn \@@_print_node_cell:
1358 { \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.

```

1359 \cs_new_protected:Npn \@@_node_for_cell:
1360 {
1361   \pgfpicture
1362   \pgfsetbaseline \c_zero_dim
1363   \pgfrememberpicturepositiononpagetrue
1364   \pgfset { nicematrix / cell-node }
1365   \pgfnode
1366   { rectangle }
1367   { base }
1368   {

```

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).

```

1369     \set@color
1370     \box_use_drop:N \l_@@_cell_box
1371   }
1372   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1373   { \l_@@_pgf_node_code_tl }
1374   \str_if_empty:NF \l_@@_name_str
1375   {
1376     \pgfnodealias
1377     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1378     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1379   }
1380   \endpgfpicture
1381 }

```

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.

```

1382 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1383 {
1384   \cs_new_protected:Npn \@@_patch_node_for_cell:
1385   {
1386     \hbox_set:Nn \l_@@_cell_box
1387     {
1388       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1389       \hbox_overlap_left:n
1390       {
1391         \pgfsys@markposition
1392         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```


In the #1, we will put an adjustment which is needed when the compilation is done with XeLaTeX or with the classical way latex, dvips, ps2pdf or Adobe Distiller (I don't know why this adjustment is mandatory...). See the use of that command \@@_patch_node_for_cell:n in a \AtBeginDocument just below.

```

1393         #1
1394     }
1395     \box_use:N \l_@@_cell_box
1396     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1397     \hbox_overlap_left:n
1398     {
1399         \pgfsys@markposition
1400         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1401         #1
1402     }
1403 }
1404 }
1405 }

```

We have no explanation for the different behaviour between the TeX engines... We put the following instructions in a \AtBeginDocument because you use \sys_if_output_div_p: and that test is available only when a backend is loaded (and we don't want to force the loading of a backend with \sys_ensure_backend:).

```

1406 \AtBeginDocument
1407 {
1408     \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1409     {
1410         \@@_patch_node_for_cell:n
1411         { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1412     }
1413     { \@@_patch_node_for_cell:n { } }
1414 }

```

The second argument of the following command \@@_instruction_of_type:nnn defined below is the type of the instruction (Cdots, Vdots, Ddots, etc.). The third argument is the list of options. This command writes in the corresponding \g_@@_type_lines_tl the instruction which will actually draw the line after the construction of the matrix.

For example, for the following matrix,

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 \\
5 & \Cdots & & 6 \\
7 & \Cdots[color=red] & & 
\end{pNiceMatrix}

```

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

the content of \g_@@_Cdots_lines_tl will be:

```

\@@_draw_Cdots:nnn {2}{2}{}
\@@_draw_Cdots:nnn {3}{2}{color=red}

```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions (with consequences for the parallelisation of the diagonal lines).

```

1415 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1416 {
1417     \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1418     { \g_@@_#2_lines_tl }
1419     {
1420         \use:c { @@_draw_#2 : nnn }
1421         { \int_use:N \c@iRow }
1422         { \int_use:N \c@jCol }
1423         { \exp_not:n { #3 } }
1424     }
1425 }

```

```

1426 \cs_generate_variant:Nn \@@_array:n { o }
1427 \cs_new_protected:Npn \@@_array:n
1428 {
1429 % \begin{macrocode}
1430 \dim_set:Nn \col@sep
1431 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1432 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1433 { \cs_set_nopar:Npn \@halignto { } }
1434 { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1435 \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).

```

1436 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1437 }

```

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.

```

1438 \bool_if:nTF
1439 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1440 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1441 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1442 \cs_new_protected:Npn \@@_create_row_node:
1443 {
1444 \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1445 {
1446 \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1447 \@@_create_row_node_i:
1448 }
1449 }
1450 \cs_new_protected:Npn \@@_create_row_node_i:
1451 {

```

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

```

1452 \hbox
1453 {
1454 \bool_if:NT \l_@@_code_before_bool
1455 {
1456 \vtop
1457 {
1458 \skip_vertical:N 0.5\arrayrulewidth
1459 \pgfsys@markposition
1460 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1461 \skip_vertical:N -0.5\arrayrulewidth
1462 }
1463 }
1464 \pgfpicture
1465 \pgfrememberpicturepositiononpagetrue
1466 \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1467 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1468 \str_if_empty:NF \l_@@_name_str
1469 {
1470 \pgfnodealias
1471 { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1472 { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1473 }

```

```

1474     \endpgfpicture
1475   }
1476 }

1477 \cs_new_protected:Npn \@@_in_everycr:
1478 {
1479   \bool_if:NT \c_@@_recent_array_bool
1480   {
1481     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1482     \tbl_update_cell_data_for_next_row:
1483   }
1484   \int_gzero:N \c@jCol
1485   \bool_gset_false:N \g_@@_after_col_zero_bool
1486   \bool_if:NF \g_@@_row_of_col_done_bool
1487   {
1488     \@@_create_row_node:

```

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

```

1489     \clist_if_empty:NF \l_@@_hlines_clist
1490     {
1491       \str_if_eq:eeF \l_@@_hlines_clist { all }
1492       {
1493         \clist_if_in:NeT
1494           \l_@@_hlines_clist
1495           { \int_eval:n { \c@iRow + 1 } }
1496       }
1497     }

```

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.

```

1498         \int_compare:nNnT \c@iRow > { -1 }
1499         {
1500           \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1501             { \hrule height \arrayrulewidth width \c_zero_dim }
1502         }
1503     }
1504 }
1505 }
1506 }

```

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

```

1507 \cs_set_protected:Npn \@@_renew_dots:
1508 {
1509   \cs_set_eq:NN \ldots \@@_Ldots
1510   \cs_set_eq:NN \cdots \@@_Cdots
1511   \cs_set_eq:NN \vdots \@@_Vdots
1512   \cs_set_eq:NN \ddots \@@_Ddots
1513   \cs_set_eq:NN \iddots \@@_Iddots
1514   \cs_set_eq:NN \dots \@@_Ldots
1515   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1516 }

```

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

```

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

```

```

1524     {
1525         \cs_new_protected:Npn \@@_everycr:
1526         { \everycr { \noalign { \@@_in_everycr: } } }
1527     }
1528 }

```

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.

```

1529 \hook_gput_code:nnn { begindocument } { . }
1530 {
1531     \IfPackageLoadedTF { booktabs }
1532     {
1533         \cs_new_protected:Npn \@@_patch_booktabs:
1534         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1535     }
1536     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1537 }

```

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`.

```

1538 \cs_new_protected:Npn \@@_some_initialization:
1539 {
1540     \@@_everycr:
1541     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1542     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1543     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1544     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1545     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1546     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1547 }

```

```

1548 \cs_new_protected:Npn \@@_pre_array_ii:
1549 {

```

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

```

1550     \int_gzero:N \g_@@_total_X_weight_int
1551     \@@_expand_clist:N \l_@@_hlines_clist
1552     \@@_expand_clist:N \l_@@_vlines_clist
1553     \@@_patch_booktabs:
1554     \box_clear_new:N \l_@@_cell_box
1555     \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}`).

```

1556     \bool_if:NT \l_@@_small_bool
1557     {

```

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

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

```

1558     \cs_set_nopar:Npn \arraystretch { 0.47 }
1559     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1560     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1561   }

1562   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1563     {
1564       \tl_put_right:Nn \@@_begin_of_row:
1565         {
1566           \pgfsys@markposition
1567             { \@@_env: - row - \int_use:N \c@iRow - base }
1568         }
1569     }

```

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`.

```

1570   \bool_if:nTF
1571     { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1572     {
1573       \cs_set_nopar:Npn \ar@ialign
1574         {
1575           \bool_if:NT \c_@@_testphase_table_bool
1576             \tbl_init_cell_data_for_table:
1577           \@@_some_initialization:
1578           \dim_zero:N \tabskip

```

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

```

1579           \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1580           \halign
1581         }
1582     }

```

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`.

```

1583     {
1584       \cs_set_nopar:Npn \ialign
1585         {
1586           \@@_some_initialization:
1587           \dim_zero:N \tabskip
1588           \cs_set_eq:NN \ialign \@@_old_ialign:
1589           \halign
1590         }
1591     }

```

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).

```

1592   \bool_if:NT \c_@@_revtex_bool
1593     {
1594       \IfPackageLoadedT { colortbl }
1595         { \cs_set_protected:Npn \CT@setup { } }
1596     }

```

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1597 \cs_set_eq:NN \@@_old_ldots \ldots
1598 \cs_set_eq:NN \@@_old_cdots \cdots
1599 \cs_set_eq:NN \@@_old_vdots \vdots
1600 \cs_set_eq:NN \@@_old_ddots \ddots
1601 \cs_set_eq:NN \@@_old_iddots \iddots
1602 \bool_if:NTF \l_@@_standard_cline_bool
1603   { \cs_set_eq:NN \cline \@@_standard_cline }
1604   { \cs_set_eq:NN \cline \@@_cline }
1605 \cs_set_eq:NN \Ldots \@@_Ldots
1606 \cs_set_eq:NN \Cdots \@@_Cdots
1607 \cs_set_eq:NN \Vdots \@@_Vdots
1608 \cs_set_eq:NN \Ddots \@@_Ddots
1609 \cs_set_eq:NN \Iddots \@@_Iddots
1610 \cs_set_eq:NN \Hline \@@_Hline:
1611 \cs_set_eq:NN \Hspace \@@_Hspace:
1612 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1613 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1614 \cs_set_eq:NN \Block \@@_Block:
1615 \cs_set_eq:NN \rotate \@@_rotate:
1616 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1617 \cs_set_eq:NN \dotfill \@@_dotfill:
1618 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1619 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1620 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1621 \cs_set_eq:NN \TopRule \@@_TopRule
1622 \cs_set_eq:NN \MidRule \@@_MidRule
1623 \cs_set_eq:NN \BottomRule \@@_BottomRule
1624 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1625 \cs_set_eq:NN \Hbrace \@@_Hbrace
1626 \cs_set_eq:NN \Vbrace \@@_Vbrace
1627 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1628   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1629 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1630 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1631 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1632 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1633 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1634   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1635 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1636   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1637 \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:`.

```

1638 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1639 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1640   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1641 \@@_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!).

```

1642 \tl_if_exist:NT \l_@@_note_in_caption_tl
1643   {
1644     \tl_if_empty:NF \l_@@_note_in_caption_tl
1645     {
1646       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl

```

```

1647         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1648     }
1649 }

```

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).

```

1650     \seq_gclear:N \g_@@_multicolumn_cells_seq
1651     \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).

```

1652     \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`).

```

1653     \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.

```

1654     \int_gzero_new:N \g_@@_col_total_int
1655     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1656     \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.

```

1657     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1658     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1659     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1660     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1661     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1662     \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

```

```

1663     \tl_gclear:N \g_nicematrix_code_before_tl
1664     \tl_gclear:N \g_@@_pre_code_before_tl
1665 }

```

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

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

```

1666 \cs_new_protected:Npn \@@_pre_array:
1667 {
1668     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1669     \int_gzero_new:N \c@iRow
1670     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1671     \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).

```

1672     \int_compare:nNnT \l_@@_last_row_int = { -1 }
1673     {
1674         \bool_set_true:N \l_@@_last_row_without_value_bool
1675         \bool_if:NT \g_@@_aux_found_bool
1676             { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1677     }

```

```

1678 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1679 {
1680   \bool_if:NT \g_@@_aux_found_bool
1681   { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1682 }

```

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”.

```

1683 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1684 {
1685   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1686   {
1687     \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1688     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1689     \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1690     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1691   }
1692 }

```

```

1693 \seq_gclear:N \g_@@_cols_vlism_seq
1694 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1695 \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.

```

1696 \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq
1697 \seq_gclear:N \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1698 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1699 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avoid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```

1700 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1701 \@@_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.

```

1702 \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]`).

```

1703 \dim_zero_new:N \l_@@_left_delim_dim
1704 \dim_zero_new:N \l_@@_right_delim_dim
1705 \bool_if:NTF \g_@@_delims_bool
1706 {

```


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

```

1707     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1708     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1709     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1710     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1711   }
1712   {
1713     \dim_gset:Nn \l_@@_left_delim_dim
1714       { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1715     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1716   }

```

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).

```

1717   \hbox_set:Nw \l_@@_the_array_box
1718   \skip_horizontal:N \l_@@_left_margin_dim
1719   \skip_horizontal:N \l_@@_extra_left_margin_dim
1720   \bool_if:NT \c_@@_recent_array_bool
1721     { \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).

```

1722   \m@th
1723   \c_math_toggle_token
1724   \bool_if:NTF \l_@@_light_syntax_bool
1725     { \use:c { @@-light-syntax } }
1726     { \use:c { @@-normal-syntax } }
1727   }

```

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

```

1728 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1729   {
1730     \tl_set:Nn \l_tmpa_tl { #1 }
1731     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1732       { \@@_rescan_for_spanish:N \l_tmpa_tl }
1733     \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1734     \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...

```

1735   \@@_pre_array:
1736   }

```

9 The `\CodeBefore`

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

```

1737 \cs_new_protected:Npn \@@_pre_code_before:
1738   {

```

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).

```

1739 \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1740 \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1741 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1742 \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.

```

1743 \pgfsys@markposition { \@@_env: - position }
1744 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1745 \pgfpicture
1746 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

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

```

Now, the recreation of the `col` nodes.

```

1753 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1754 {
1755   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1756   \pgfcoordinate { \@@_env: - col - ##1 }
1757   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1758 }

```

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

```

1759 \@@_create_diag_nodes:

```

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

```

1760 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1761 \endpgfpicture

```

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

```

1762 \@@_create_blocks_nodes:
1763 \IfPackageLoadedT { tikz }
1764 {
1765   \tikzset
1766   {
1767     every-picture / .style =
1768     { overlay , name-prefix = \@@_env: - }
1769   }
1770 }
1771 \cs_set_eq:NN \cellcolor \@@_cellcolor
1772 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1773 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1774 \cs_set_eq:NN \rowcolor \@@_rowcolor
1775 \cs_set_eq:NN \rowcolors \@@_rowcolors
1776 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1777 \cs_set_eq:NN \arraycolor \@@_arraycolor
1778 \cs_set_eq:NN \columncolor \@@_columncolor
1779 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1780 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1781 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1782 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1783 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n

```

```

1784 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1785 }

```

```

1786 \cs_new_protected:Npn \@@_exec_code_before:
1787 {

```

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...

```

1788 \clist_map_inline:Nn \l_@@_corners_cells_clist
1789 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1790 \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`.

```

1791 \@@_add_to_colors_seq:n { { nocolor } } { }
1792 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1793 \group_begin:

```

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

```

1794 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token

```

The following code is a security for the case the user has used `babel` with the option `spanish`: in that case, the characters `<` (de code ASCII 60) and `>` are activated and Tikz is not able to solve the problem (even with the Tikz library `babel`).

```

1795 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1796 { \@@_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`.

```

1797 \exp_last_unbraced:No \@@_CodeBefore_keys:
1798 \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.

```

1799 \@@_actually_color:
1800 \l_@@_code_before_tl
1801 \q_stop
1802 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1803 \group_end:
1804 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1805 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1806 }

```

```

1807 \keys_define:nm { nicematrix / CodeBefore }
1808 {
1809 create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1810 create-cell-nodes .default:n = true ,
1811 sub-matrix .code:n = \keys_set:nm { nicematrix / sub-matrix } { #1 } ,
1812 sub-matrix .value_required:n = true ,
1813 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1814 delimiters / color .value_required:n = true ,
1815 unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1816 }

```

```

1817 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1818 {
1819   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1820   \@@_CodeBefore:w
1821 }

```

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.

```

1822 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1823 {
1824   \bool_if:NT \g_@@_aux_found_bool
1825   {
1826     \@@_pre_code_before:
1827     \legacy_if:nF { measuring@ } { #1 }
1828   }
1829 }

```

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.

```

1830 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1831 {
1832   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1833   {
1834     \pgfsys@getposition { \@@_env: - #1 - base } \@@_node_position:
1835     \pgfcoordinate { \@@_env: - row - #1 - base }
1836     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1837     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1838     {
1839       \cs_if_exist:cT
1840       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - #1 - #####1 - NW }
1841       {
1842         \pgfsys@getposition
1843         { \@@_env: - #1 - #####1 - NW }
1844         \@@_node_position:
1845         \pgfsys@getposition
1846         { \@@_env: - #1 - #####1 - SE }
1847         \@@_node_position_i:
1848         \@@_pgf_rect_node:nnn
1849         { \@@_env: - #1 - #####1 }
1850         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1851         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1852       }
1853     }
1854   }
1855   \int_step_inline:nn \c@iRow
1856   {
1857     \pgfnodealias
1858     { \@@_env: - #1 - last }
1859     { \@@_env: - #1 - \int_use:N \c@jCol }
1860   }
1861   \int_step_inline:nn \c@jCol
1862   {
1863     \pgfnodealias
1864     { \@@_env: - last - #1 }
1865     { \@@_env: - \int_use:N \c@iRow - #1 }
1866   }
1867   \@@_create_extra_nodes:
1868 }

```

```

1869 \cs_new_protected:Npn \@@_create_blocks_nodes:
1870 {
1871   \pgfpicture
1872   \pgf@relevantforpicturesizefalse
1873   \pgfrememberpicturepositiononpagetrue
1874   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1875     { \@@_create_one_block_node:nnnnn ##1 }
1876   \endpgfpicture
1877 }

```

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

```

1878 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1879 {
1880   \tl_if_empty:nF { #5 }
1881   {
1882     \@@_qpoint:n { col - #2 }
1883     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1884     \@@_qpoint:n { #1 }
1885     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1886     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1887     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1888     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1889     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1890     \@@_pgf_rect_node:nnnnn
1891       { \@@_env: - #5 }
1892       { \dim_use:N \l_tmpa_dim }
1893       { \dim_use:N \l_tmpb_dim }
1894       { \dim_use:N \l_@@_tmpc_dim }
1895       { \dim_use:N \l_@@_tmpd_dim }
1896   }
1897 }

```

```

1898 \cs_new_protected:Npn \@@_patch_for_revtext:
1899 {
1900   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1901   \cs_set_eq:NN \@array \@array@array
1902   \cs_set_eq:NN \@tabular \@tabular@array
1903   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1904   \cs_set_eq:NN \array \array@array
1905   \cs_set_eq:NN \endarray \endarray@array
1906   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1907   \cs_set_eq:NN \@mkpream \@mkpream@array
1908   \cs_set_eq:NN \@classx \@classx@array
1909   \cs_set_eq:NN \insert@column \insert@column@array
1910   \cs_set_eq:NN \@arraycr \@arraycr@array
1911   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1912   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1913 }

```

10 The environment `{NiceArrayWithDelims}`

```

1914 \NewDocumentEnvironment { NiceArrayWithDelims }
1915   { m m 0 { } m ! 0 { } t \CodeBefore }
1916   {

```

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

```

1917 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1918 \@@_provide_pgfsyspdfmark:
1919 \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.

```

1920 \bgroup

1921 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1922 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1923 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1924 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1925 \int_gzero:N \g_@@_block_box_int
1926 \dim_zero:N \g_@@_width_last_col_dim
1927 \dim_zero:N \g_@@_width_first_col_dim
1928 \bool_gset_false:N \g_@@_row_of_col_done_bool
1929 \str_if_empty:NT \g_@@_name_env_str
1930 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1931 \bool_if:NTF \l_@@_tabular_bool
1932 \mode_leave_vertical:
1933 \@@_test_if_math_mode:
1934 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1935 \bool_set_true:N \l_@@_in_env_bool

```

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

```

1936 \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 `\tikzexternalisable` and not with `\tikzset{external/export=false}` which is *not* equivalent.

```

1937 \cs_if_exist:NT \tikz@library@external@loaded
1938 {
1939 \tikzexternalisable
1940 \cs_if_exist:NT \ifstandalone
1941 { \tikzset { external / optimize = false } }
1942 }

```

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

```

1943 \int_gincr:N \g_@@_env_int
1944 \bool_if:NF \l_@@_block_auto_columns_width_bool
1945 { \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.

```

1946 \seq_gclear:N \g_@@_blocks_seq
1947 \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`.

```

1948 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1949 \seq_gclear:N \g_@@_pos_of_xdots_seq
1950 \tl_gclear_new:N \g_@@_code_before_tl
1951 \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.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1952 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1953 {
1954   \bool_gset_true:N \g_@@_aux_found_bool
1955   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1956 }
1957 { \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.

```

1958 \tl_gclear:N \g_@@_aux_tl
1959 \tl_if_empty:NF \g_@@_code_before_tl
1960 {
1961   \bool_set_true:N \l_@@_code_before_bool
1962   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1963 }
1964 \tl_if_empty:NF \g_@@_pre_code_before_tl
1965 { \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`.

```

1966 \bool_if:NTF \g_@@_delims_bool
1967 { \keys_set:nn { nicematrix / pNiceArray } }
1968 { \keys_set:nn { nicematrix / NiceArray } }
1969 { #3 , #5 }

```

```

1970 \@@_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:`.

```

1971 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1972 }

```

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

```

1973 {
1974   \bool_if:NTF \l_@@_light_syntax_bool
1975   { \use:c { end @@-light-syntax } }
1976   { \use:c { end @@-normal-syntax } }
1977   \c_math_toggle_token
1978   \skip_horizontal:N \l_@@_right_margin_dim
1979   \skip_horizontal:N \l_@@_extra_right_margin_dim
1980
1981   % awful workaround
1982   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1983   {
1984     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1985     {
1986       \skip_horizontal:N - \l_@@_columns_width_dim
1987       \bool_if:NTF \l_@@_tabular_bool
1988       { \skip_horizontal:n { - 2 \tabcolsep } }
1989       { \skip_horizontal:n { - 2 \arraycolsep } }
1990     }
1991   }
1992   \hbox_set_end:
1993   \bool_if:NT \c_@@_recent_array_bool
1994   { \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.

```

1995   \bool_if:NT \l_@@_width_used_bool
1996   {
1997     \int_if_zero:nT \g_@@_total_X_weight_int
1998     { \@@_error_or_warning:n { width~without~X~columns } }
1999   }

```

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 .

```

2000   \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
2001   { \@@_compute_width_X: }

```

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

```

2002   \int_compare:nNnT \l_@@_last_row_int > { -2 }
2003   {
2004     \bool_if:NF \l_@@_last_row_without_value_bool
2005     {
2006       \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2007       {
2008         \@@_error:n { Wrong~last~row }
2009         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2010       }
2011     }
2012   }

```

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

```

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

```

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

```

2020   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2021   \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 90).

```

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

```

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

```

2024   \bool_if:nTF { ! \g_@@_delims_bool }
2025   {
2026     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2027     \@@_use_arraybox_with_notes_c:
2028     {
2029       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2030       \@@_use_arraybox_with_notes_b:
2031       \@@_use_arraybox_with_notes:
2032     }
2033   }

```

Now, in the case of an environment with delimiters. We compute `\l_tmpa_dim` which is the total height of the “first row” above the array (when the key `first-row` is used).

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


```

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

```

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

```

2041   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2042   {
2043     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2044     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2045   }
2046   { \dim_zero:N \l_tmpb_dim }
2047   \hbox_set:Nn \l_tmpa_box
2048   {
2049     \m@th % added 2024/11/21
2050     \c_math_toggle_token
2051     \@@_color:o \l_@@_delimiters_color_tl
2052     \exp_after:wN \left \g_@@_left_delim_tl
2053     \vcenter
2054     {

```

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.

```

2055       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2056       \hbox
2057       {
2058         \bool_if:NTF \l_@@_tabular_bool
2059         { \skip_horizontal:N -\tabcolsep }
2060         { \skip_horizontal:N -\arraycolsep }
2061         \@@_use_arraybox_with_notes_c:
2062         \bool_if:NTF \l_@@_tabular_bool
2063         { \skip_horizontal:N -\tabcolsep }
2064         { \skip_horizontal:N -\arraycolsep }
2065       }

```

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

```

2066       \skip_vertical:N -\l_tmpb_dim
2067       \skip_vertical:N \arrayrulewidth
2068     }
2069     \exp_after:wN \right \g_@@_right_delim_tl
2070     \c_math_toggle_token
2071   }

```

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.

```

2072   \bool_if:NTF \l_@@_delimiters_max_width_bool
2073   {
2074     \@@_put_box_in_flow_bis:nn
2075     \g_@@_left_delim_tl
2076     \g_@@_right_delim_tl
2077   }
2078   \@@_put_box_in_flow:
2079 }

```

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

```

2080   \bool_if:NT \g_@@_last_col_found_bool

```

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

```

2081     { \skip_horizontal:N \g_@@_width_last_col_dim }
2082 \bool_if:NT \l_@@_preamble_bool
2083   {
2084     \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2085     { \@@_warning_gredirect_none:n { columns-not-used } }
2086   }
2087 \@@_after_array:

```

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

```

2088 \egroup

```

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

```

2089 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2090 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2091 \iow_now:Ne \@mainaux
2092   {
2093     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2094     { \exp_not:o \g_@@_aux_tl }
2095   }
2096 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2097 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2098 }

```

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 .

```

2099 \cs_new_protected:Npn \@@_compute_width_X:
2100   {
2101     \tl_gput_right:Ne \g_@@_aux_tl
2102     {
2103       \bool_set_true:N \l_@@_X_columns_aux_bool
2104       \dim_set:Nn \l_@@_X_columns_dim
2105       {
2106         \dim_compare:nNnTF
2107         {
2108           \dim_abs:n
2109           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2110         }
2111         <
2112         { 0.001 pt }
2113         { \dim_use:N \l_@@_X_columns_dim }
2114         {
2115           \dim_eval:n
2116           {
2117             ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2118             / \int_use:N \g_@@_total_X_weight_int
2119             + \l_@@_X_columns_dim
2120           }
2121         }
2122       }
2123     }
2124 }

```

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.

```

2125 \cs_new_protected:Npn \@@_transform_preamble:
2126 {
2127   \@@_transform_preamble_i:
2128   \@@_transform_preamble_ii:
2129 }

2130 \cs_new_protected:Npn \@@_transform_preamble_i:
2131 {
2132   \int_gzero:N \c@jCol

```

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

```

2133   \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.

```

2134   \bool_gset_false:N \g_tmpb_bool

```

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

```

2135   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2136   \int_zero:N \l_tmpa_int
2137   \tl_gclear:N \g_@@_array_preamble_tl
2138   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2139   {
2140     \tl_gset:Nn \g_@@_array_preamble_tl
2141     { ! { \skip_horizontal:N \arrayrulewidth } }
2142   }
2143   {
2144     \clist_if_in:NnT \l_@@_vlines_clist 1
2145     {
2146       \tl_gset:Nn \g_@@_array_preamble_tl
2147       { ! { \skip_horizontal:N \arrayrulewidth } }
2148     }
2149   }

```

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

```

2150   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2151   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2152   \@@_replace_columncolor:
2153 }

```

```

2154 \hook_gput_code:nnn { begindocument } { . }
2155 {
2156   \IfPackageLoadedTF { colortbl }
2157   {

```

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).

```

2158     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2159     \cs_new_protected:Npn \@@_replace_columncolor:
2160     {
2161         \regex_replace_all:NnN
2162         \c_@@_columncolor_regex
2163         { \c { @@_columncolor_preamble } }
2164         \g_@@_array_preamble_tl
2165     }
2166 }
2167 {
2168     \cs_new_protected:Npn \@@_replace_columncolor:
2169     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2170 }
2171 }

```

```

2172 \cs_new_protected:Npn \@@_transform_preamble_ii:
2173 {

```

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

```

2174     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2175     {
2176         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2177         { \bool_gset_true:N \g_@@_delims_bool }
2178     }
2179     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2180     \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).

```

2181     \int_if_zero:nTF \l_@@_first_col_int
2182     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2183     {
2184         \bool_if:NF \g_@@_delims_bool
2185         {
2186             \bool_if:NF \l_@@_tabular_bool
2187             {
2188                 \clist_if_empty:NT \l_@@_vlines_clist
2189                 {
2190                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2191                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2192                 }
2193             }
2194         }
2195     }
2196     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2197     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2198     {
2199         \bool_if:NF \g_@@_delims_bool
2200         {
2201             \bool_if:NF \l_@@_tabular_bool
2202             {
2203                 \clist_if_empty:NT \l_@@_vlines_clist
2204                 {
2205                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2206                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2207                 }
2208             }
2209         }

```

```

2208     }
2209   }
2210 }

```

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`).

```

2211 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2212 {

```

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.

```

2213   \bool_if:NF \c_@@_testphase_table_bool
2214   {
2215     \tl_gput_right:Nn \g_@@_array_preamble_tl
2216     { > { \@@_error_too_much_cols: } l }
2217   }
2218 }
2219 }

```

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.

```

2220 \cs_new_protected:Npn \@@_rec_preamble:n #1
2221 {

```

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

```

2222   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2223   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2224   {

```

Now, the columns defined by `\newcolumn` type of `array`.

```

2225     \cs_if_exist:cTF { NC @ find @ #1 }
2226     {
2227       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2228       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2229     }
2230     {
2231       \str_if_eq:nnTF { #1 } { S }
2232       { \@@_fatal:n { unknown~column~type~S } }
2233       { \@@_fatal:nn { unknown~column~type } { #1 } }
2234     }
2235   }
2236 }

```

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

```

2237 \cs_new_protected:Npn \@@_c #1
2238 {
2239   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2240   \tl_gclear:N \g_@@_pre_cell_tl
2241   \tl_gput_right:Nn \g_@@_array_preamble_tl
2242   { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2243   \int_gincr:N \c@jCol
2244   \@@_rec_preamble_after_col:n
2245 }

```

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

```

2246 \cs_new_protected:Npn \@@_l #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_@@_l_tl }
2253     l
2254     < \@@_cell_end:
2255   }
2256   \int_gincr:N \c@jCol
2257   \@@_rec_preamble_after_col:n
2258 }

2259 \cs_new_protected:Npn \@@_r #1
2260 {
2261   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2262   \tl_gclear:N \g_@@_pre_cell_tl
2263   \tl_gput_right:Nn \g_@@_array_preamble_tl
2264   {
2265     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2266     r
2267     < \@@_cell_end:
2268   }
2269   \int_gincr:N \c@jCol
2270   \@@_rec_preamble_after_col:n
2271 }

```

For ! and @

```

2272 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2273 {
2274   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2275   \@@_rec_preamble:n
2276 }
2277 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2278 \cs_new_protected:cpn { @@ _ | } #1
2279 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2280   \int_incr:N \l_tmpa_int
2281   \@@_make_preamble_i_i:n
2282 }

2283 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2284 {
2285   \str_if_eq:nnTF { #1 } { | }
2286   { \use:c { @@ _ | } | }
2287   { \@@_make_preamble_i_ii:nn { } #1 }
2288 }

2289 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2290 {
2291   \str_if_eq:nnTF { #2 } { [ ]
2292   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2293   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2294 }
2295 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2296 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2297 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2298 {
2299   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2300   \tl_gput_right:Ne \g_@@_array_preamble_tl
2301   {

```

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

```

2302     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2303   }
2304   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2305   {
2306     \@@_vline:n
2307     {
2308       position = \int_eval:n { \c@jCol + 1 } ,
2309       multiplicity = \int_use:N \l_tmpa_int ,
2310       total-width = \dim_use:N \l_@@_rule_width_dim ,
2311       #2
2312     }

```

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.

```

2313   }
2314   \int_zero:N \l_tmpa_int
2315   \str_if_eq:nnF { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2316   \@@_rec_preamble:n #1
2317 }

2318 \cs_new_protected:cpn { @@ _ > } #1 #2
2319 {
2320   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2321   \@@_rec_preamble:n
2322 }

2323 \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.

```

2324 \keys_define:nn { nicematrix / p-column }
2325 {
2326   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2327   r .value_forbidden:n = true ,
2328   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2329   c .value_forbidden:n = true ,
2330   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2331   l .value_forbidden:n = true ,
2332   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2333   S .value_forbidden:n = true ,
2334   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2335   p .value_forbidden:n = true ,
2336   t .meta:n = p ,
2337   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2338   m .value_forbidden:n = true ,
2339   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2340   b .value_forbidden:n = true
2341 }

```

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

```

2342 \cs_new_protected:Npn \@@_p #1
2343 {
2344   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2345   \@@_make_preamble_ii_i:n
2346 }
2347 \cs_set_eq:NN \@@_b \@@_p
2348 \cs_set_eq:NN \@@_m \@@_p

```

```

2349 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2350 {
2351   \str_if_eq:nnTF { #1 } { [ ]
2352     { \@@_make_preamble_ii_ii:w [ ]
2353       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2354     }
2355   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2356     { \@@_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.

```

2357 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2358 {

```

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).

```

2359   \str_set:Nn \l_@@_hpos_col_str { j }
2360   \@@_keys_p_column:n { #1 }
2361   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2362 }
2363 \cs_new_protected:Npn \@@_keys_p_column:n #1
2364 { \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.

```

2365 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2366 {
2367   \use:e
2368   {
2369     \@@_make_preamble_ii_v:nnnnnnn
2370     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2371     { \dim_eval:n { #1 } }
2372     {

```

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.

```

2373     \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2374     { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2375     {

```

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

```

2376     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2377     { \str_lowercase:o \l_@@_hpos_col_str }
2378   }
2379   \IfPackageLoadedTF { ragged2e }
2380   {
2381     \str_case:on \l_@@_hpos_col_str
2382     {
2383       c { \exp_not:N \Centering }
2384       l { \exp_not:N \RaggedRight }
2385       r { \exp_not:N \RaggedLeft }
2386     }
2387   }
2388   {
2389     \str_case:on \l_@@_hpos_col_str
2390     {
2391       c { \exp_not:N \centering }
2392       l { \exp_not:N \raggedright }
2393       r { \exp_not:N \raggedleft }
2394     }
2395   }

```



```

2396         #3
2397     }
2398     { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2399     { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2400     { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2401     { #2 }
2402     {
2403         \str_case:onF \l_@@_hpos_col_str
2404         {
2405             { j } { c }
2406             { si } { c }
2407         }

```

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

```

2408         { \str_lowercase:o \l_@@_hpos_col_str }
2409     }
2410 }

```

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

```

2411     \int_gincr:N \c@jCol
2412     \@@_rec_preamble_after_col:n
2413 }

```

#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*.

```

2414 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2415 {
2416     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2417     {
2418         \tl_gput_right:Nn \g_@@_array_preamble_tl
2419         { > \@@_test_if_empty_for_S: }
2420     }
2421     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2422     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2423     \tl_gclear:N \g_@@_pre_cell_tl
2424     \tl_gput_right:Nn \g_@@_array_preamble_tl
2425     {
2426         > {

```

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.

```

2427         \dim_set:Nn \l_@@_col_width_dim { #2 }
2428         \bool_if:NT \c_@@_testphase_table_bool
2429         { \tag_struct_begin:n { tag = Div } }
2430         \@@_cell_begin:

```

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

```

2431         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2432     \everypar
2433     {
2434         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2435         \everypar { }
2436     }
2437     \bool_if:NT \c_@@_testphase_table_bool \tagpdfpara0n

```

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

```

2438     #3

```

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

```

2439     \g_@@_row_style_tl
2440     \arraybackslash
2441     #5
2442     }
2443     #8
2444     < {
2445     #6

```

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

```

2446     \@finalstrut \@arstrutbox
2447     \use:c { end #7 }

```

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

```

2448     #4
2449     \@@_cell_end:
2450     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2451     }
2452     }
2453     }

```

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

```

2454 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2455 {

```

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

```

2456 \group_align_safe_begin:
2457 \peek_meaning:NTF &
2458 \@@_the_cell_is_empty:
2459 {
2460     \peek_meaning:NTF \\\
2461     \@@_the_cell_is_empty:
2462     {
2463         \peek_meaning:NTF \crcr
2464         \@@_the_cell_is_empty:
2465         \group_align_safe_end:
2466     }
2467 }
2468 }

2469 \cs_new_protected:Npn \@@_the_cell_is_empty:
2470 {
2471     \group_align_safe_end:
2472     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2473     {

```

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

```

2474     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2475     \skip_horizontal:N \l_@@_col_width_dim

```

```

2476     }
2477 }
2478 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2479 {
2480   \peek_meaning:NT \_siunitx_table_skip:n
2481   { \bool_gset_true:N \g_@@_empty_cell_bool }
2482 }

```

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.

```

2483 \cs_new_protected:Npn \@@_center_cell_box:
2484 {

```

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

```

2485   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2486   {
2487     \int_compare:nNnT
2488     { \box_ht:N \l_@@_cell_box }
2489     >

```

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).

```

2490     { \box_ht:N \strutbox }
2491     {
2492       \hbox_set:Nn \l_@@_cell_box
2493       {
2494         \box_move_down:nn
2495         {
2496           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2497             + \baselineskip ) / 2
2498         }
2499         { \box_use:N \l_@@_cell_box }
2500       }
2501     }
2502 }
2503 }

```

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

```

2504 \cs_new_protected:Npn \@@_V #1 #2
2505 {
2506   \str_if_eq:nnTF { #1 } { [ ]
2507     { \@@_make_preamble_V_i:w [ ]
2508       { \@@_make_preamble_V_i:w [ ] { #2 } }
2509     }
2510   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2511     { \@@_make_preamble_V_ii:nn { #1 } }
2512   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2513     {
2514       \str_set:Nn \l_@@_vpos_col_str { p }
2515       \str_set:Nn \l_@@_hpos_col_str { j }
2516       \@@_keys_p_column:n { #1 }
2517       \IfPackageLoadedTF { varwidth }
2518       { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2519       {
2520         \@@_error_or_warning:n { varwidth-not-loaded }
2521         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2522       }
2523     }

```

For w and W

```

2524 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2525 \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.

```

2526 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2527 {
2528   \str_if_eq:nnTF { #3 } { s }
2529   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2530   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2531 }

```

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.

```

2532 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2533 {
2534   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2535   \tl_gclear:N \g_@@_pre_cell_tl
2536   \tl_gput_right:Nn \g_@@_array_preamble_tl
2537   {
2538     > {
2539       \dim_set:Nn \l_@@_col_width_dim { #2 }
2540       \@@_cell_begin:
2541       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2542     }
2543     c
2544     < {
2545       \@@_cell_end_for_w_s:
2546       #1
2547       \@@_adjust_size_box:
2548       \box_use_drop:N \l_@@_cell_box
2549     }
2550   }
2551   \int_gincr:N \c@jCol
2552   \@@_rec_preamble_after_col:n
2553 }

```

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

```

2554 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2555 {
2556   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2557   \tl_gclear:N \g_@@_pre_cell_tl
2558   \tl_gput_right:Nn \g_@@_array_preamble_tl
2559   {
2560     > {

```

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.

```

2561       \dim_set:Nn \l_@@_col_width_dim { #4 }
2562       \hbox_set:Nw \l_@@_cell_box
2563       \@@_cell_begin:
2564       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2565     }
2566     c
2567     < {
2568       \@@_cell_end:
2569       \hbox_set_end:
2570       #1

```

```

2571         \@@_adjust_size_box:
2572         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2573     }
2574 }

```

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

```

2575     \int_gincr:N \c@jCol
2576     \@@_rec_preamble_after_col:n
2577 }

```

```

2578 \cs_new_protected:Npn \@@_special_W:
2579 {
2580     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2581     { \@@_warning:n { W~warning } }
2582 }

```

For S (of siunitx).

```

2583 \cs_new_protected:Npn \@@_S #1 #2
2584 {
2585     \str_if_eq:nnTF { #2 } { [ ]
2586         { \@@_make_preamble_S:w [ ] }
2587         { \@@_make_preamble_S:w [ ] { #2 } }
2588     }
2589     \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2590     { \@@_make_preamble_S_i:n { #1 } }
2591     \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2592     {
2593         \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx-not-loaded } }
2594         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2595         \tl_gcclear:N \g_@@_pre_cell_tl
2596         \tl_gput_right:Nn \g_@@_array_preamble_tl
2597         {
2598             > {

```

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

```

2599         \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2600         \keys_set:nn { siunitx } { #1 }
2601         \@@_cell_begin:
2602         \siunitx_cell_begin:w
2603     }
2604     c
2605     <
2606     {
2607         \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`).

```

2608         \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2609         {
2610             \bool_if:NTF \l__siunitx_table_text_bool
2611                 \bool_set_true:N
2612                 \bool_set_false:N
2613             \l__siunitx_table_text_bool
2614         }
2615         \@@_cell_end:
2616     }
2617 }

```

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

```

2618     \int_gincr:N \c@jCol
2619     \@@_rec_preamble_after_col:n
2620 }

```

For (, [and \{.

```

2621 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2622 {
2623     \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.

```

2624     \int_if_zero:nTF \c@jCol
2625     {
2626         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2627         {

```

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

```

2628             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2629             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2630             \@@_rec_preamble:n #2
2631         }
2632     {
2633         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2634         \@@_make_preamble_iv:nn { #1 } { #2 }
2635     }
2636 }
2637 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2638 }
2639 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2640 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2641 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2642 {
2643     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2644     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2645     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2646     {
2647         \@@_error:nn { delimiter~after~opening } { #2 }
2648         \@@_rec_preamble:n
2649     }
2650     { \@@_rec_preamble:n #2 }
2651 }

```

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

```

2652 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2653 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2654 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2655 {
2656     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2657     \tl_if_in:nnTF { ) ] \} } { #2 }
2658     { \@@_make_preamble_v:nnn #1 #2 }
2659     {
2660         \str_if_eq:nnTF { \@@_stop: } { #2 }
2661         {
2662             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2663             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2664             {
2665                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2666         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2667         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2668         \@@_rec_preamble:n #2
2669     }
2670 }
2671 {
2672     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2673     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2674     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2675     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2676     \@@_rec_preamble:n #2
2677 }
2678 }
2679 }
2680 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2681 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2682 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2683 {
2684     \str_if_eq:nnTF { \@@_stop: } { #3 }
2685     {
2686         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2687         {
2688             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2689             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2690             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2691             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2692         }
2693         {
2694             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2695             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2696             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2697             \@@_error:nn { double~closing~delimiter } { #2 }
2698         }
2699     }
2700     {
2701         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2702         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2703         \@@_error:nn { double~closing~delimiter } { #2 }
2704         \@@_rec_preamble:n #3
2705     }
2706 }
2707 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2708 { \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 @{...}.

```

2709 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2710 {
2711     \str_if_eq:nnTF { #1 } { < }
2712     \@@_rec_preamble_after_col_i:n
2713     {
2714         \str_if_eq:nnTF { #1 } { @ }
2715         \@@_rec_preamble_after_col_ii:n
2716         {
2717             \str_if_eq:eeTF \l_@@_vlines_clist { all }
2718             {
2719                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2720                 { ! { \skip_horizontal:N \arrayrulewidth } }
2721             }
2722         }

```

```

2723         \clist_if_in:NeT \l_@@_vlines_clist
2724         { \int_eval:n { \c@jCol + 1 } }
2725         {
2726         \tl_gput_right:Nn \g_@@_array_preamble_tl
2727         { ! { \skip_horizontal:N \arrayrulewidth } }
2728         }
2729     }
2730     \@@_rec_preamble:n { #1 }
2731 }
2732 }
2733 }
2734 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2735 {
2736 \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2737 \@@_rec_preamble_after_col:n
2738 }

```

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.

```

2739 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2740 {
2741 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2742 {
2743 \tl_gput_right:Nn \g_@@_array_preamble_tl
2744 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2745 }
2746 {
2747 \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2748 {
2749 \tl_gput_right:Nn \g_@@_array_preamble_tl
2750 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2751 }
2752 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2753 }
2754 \@@_rec_preamble:n
2755 }
2756 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2757 {
2758 \tl_clear:N \l_tmpa_tl
2759 \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2760 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2761 }

```

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

```

2762 \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.

```

2763 \cs_new_protected:Npn \@@_X #1 #2
2764 {
2765 \str_if_eq:nnTF { #2 } { [ ]
2766 { \@@_make_preamble_X:w [ ]
2767 { \@@_make_preamble_X:w [ ] #2 }
2768 }
2769 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2770 { \@@_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`).

```
2771 \keys_define:nn { nicematrix / X-column }
2772 { 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.

```
2773 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2774 {
```

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).

```
2775 \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).

```
2776 \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`.

```
2777 \int_zero_new:N \l_@@_weight_int
2778 \int_set_eq:NN \l_@@_weight_int \c_one_int
2779 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2780 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2781 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2782 {
2783   \@@_error_or_warning:n { negative-weight }
2784   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2785 }
2786 \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).

```
2787 \bool_if:NTF \l_@@_X_columns_aux_bool
2788 {
2789   \@@_make_preamble_ii_iv:nnn
2790   { \l_@@_weight_int \l_@@_X_columns_dim }
2791   { minipage }
2792   { \@@_no_update_width: }
2793 }
2794 {
2795   \tl_gput_right:Nn \g_@@_array_preamble_tl
2796   {
2797     > {
2798       \@@_cell_begin:
2799       \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`.

```
2800 \NotEmpty
```

The following code will nullify the box of the cell.

```
2801 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2802 { \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`.

```

2803         \begin { minipage } { 5 cm } \arraybackslash
2804     }
2805     c
2806     < {
2807         \end { minipage }
2808         \@@_cell_end:
2809     }
2810 }
2811 \int_gincr:N \c@jCol
2812 \@@_rec_preamble_after_col:n
2813 }
2814 }

2815 \cs_new_protected:Npn \@@_no_update_width:
2816 {
2817     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2818     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2819 }

```

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

```

2820 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2821 {
2822     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2823     { \int_eval:n { \c@jCol + 1 } }
2824     \tl_gput_right:Ne \g_@@_array_preamble_tl
2825     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2826     \@@_rec_preamble:n
2827 }

```

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.

```

2828 \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).

```

2829 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2830 { \@@_fatal:n { Preamble-forgotten } }
2831 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2832 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2833 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2834 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2835 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2836 \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).

```

2837 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2838 {

```

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

```

2839     \multispan { #1 }
2840     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:

```

```

2841 \begingroup
2842 \bool_if:NT \c_@@_testphase_table_bool
2843   { \tbl_update_multicolumn_cell_data:n { #1 } }
2844 \cs_set_nopar:Npn \@addamp
2845   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2846 \tl_gclear:N \g_@@_preamble_tl
2847 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2848 \exp_args:No \@mkpream \g_@@_preamble_tl
2849 \@addtopreamble \@empty
2850 \endgroup
2851 \bool_if:NT \c_@@_recent_array_bool
2852   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2853 \int_compare:nNnT { #1 } > \c_one_int
2854 {
2855   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2856     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2857   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2858   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2859     {
2860       {
2861         \int_if_zero:nTF \c@jCol
2862           { \int_eval:n { \c@iRow + 1 } }
2863           { \int_use:N \c@iRow }
2864       }
2865       { \int_eval:n { \c@jCol + 1 } }
2866     {
2867       \int_if_zero:nTF \c@jCol
2868         { \int_eval:n { \c@iRow + 1 } }
2869         { \int_use:N \c@iRow }
2870     }
2871     { \int_eval:n { \c@jCol + #1 } }

```

The last argument is for the name of the block

```

2872     { }
2873   }
2874 }

```

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

```

2875 \RenewDocumentCommand \cellcolor { 0 { } m }
2876 {
2877   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2878     {
2879       \@@_rectanglecolor [ ##1 ]
2880       { \exp_not:n { ##2 } }
2881       { \int_use:N \c@iRow - \int_use:N \c@jCol }
2882       { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2883     }
2884   \ignorespaces
2885 }

```

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

```

2886 \cs_set_nopar:Npn \@sharp { #3 }
2887 \@arstrut
2888 \@preamble
2889 \null

```

We add some lines.

```

2890     \int_gadd:Nn \c@jCol { #1 - 1 }
2891     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2892         { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2893     \ignorespaces
2894 }

```

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`.

```

2895 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2896 {
2897     \str_case:nnF { #1 }
2898     {
2899         c { \@@_make_m_preamble_i:n #1 }
2900         l { \@@_make_m_preamble_i:n #1 }
2901         r { \@@_make_m_preamble_i:n #1 }
2902         > { \@@_make_m_preamble_ii:nn #1 }
2903         ! { \@@_make_m_preamble_ii:nn #1 }
2904         @ { \@@_make_m_preamble_ii:nn #1 }
2905         | { \@@_make_m_preamble_iii:n #1 }
2906         p { \@@_make_m_preamble_iv:nnn t #1 }
2907         m { \@@_make_m_preamble_iv:nnn c #1 }
2908         b { \@@_make_m_preamble_iv:nnn b #1 }
2909         w { \@@_make_m_preamble_v:nnnn { } #1 }
2910         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2911         \q_stop { }
2912     }
2913     {
2914         \cs_if_exist:cTF { NC @ find @ #1 }
2915         {
2916             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2917             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2918         }
2919         {
2920             \str_if_eq:nnTF { #1 } { S }
2921             { \@@_fatal:n { unknown~column~type~S } }
2922             { \@@_fatal:nn { unknown~column~type } { #1 } }
2923         }
2924     }
2925 }

```

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

```

2926 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2927 {
2928     \tl_gput_right:Nn \g_@@_preamble_tl
2929     {
2930         > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2931         #1
2932         < \@@_cell_end:
2933     }

```

We test for the presence of a `<`.

```

2934     \@@_make_m_preamble_x:n
2935 }

```

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

```

2936 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2937 {
2938     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2939     \@@_make_m_preamble:n
2940 }

```

For l

```
2941 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2942 {
2943   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2944   \@@_make_m_preamble:n
2945 }
```

For p, m and b

```
2946 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2947 {
2948   \tl_gput_right:Nn \g_@@_preamble_tl
2949   {
2950     > {
2951       \@@_cell_begin:
2952       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2953       \mode_leave_vertical:
2954       \arraybackslash
2955       \vrule height \box_ht:N \@@_arstrutbox depth 0 pt width 0 pt
2956     }
2957     c
2958     < {
2959       \vrule height 0 pt depth \box_dp:N \@@_arstrutbox width 0 pt
2960       \end { minipage }
2961       \@@_cell_end:
2962     }
2963   }
2964 }
```

We test for the presence of a <.

```
2964 \@@_make_m_preamble_x:n
2965 }
```

For w and W

```
2966 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2967 {
2968   \tl_gput_right:Nn \g_@@_preamble_tl
2969   {
2970     > {
2971       \dim_set:Nn \l_@@_col_width_dim { #4 }
2972       \hbox_set:Nw \l_@@_cell_box
2973       \@@_cell_begin:
2974       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2975     }
2976     c
2977     < {
2978       \@@_cell_end:
2979       \hbox_set_end:
2980       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2981       #1
2982       \@@_adjust_size_box:
2983       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2984     }
2985   }
2986 }
```

We test for the presence of a <.

```
2986 \@@_make_m_preamble_x:n
2987 }
```

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

```
2988 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2989 {
2990   \str_if_eq:nnTF { #1 } { < }
2991   \@@_make_m_preamble_ix:n
2992   { \@@_make_m_preamble:n { #1 } }
2993 }
```

```

2994 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2995 {
2996   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2997   \@@_make_m_preamble_x:n
2998 }

```

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`).

```

2999 \cs_new_protected:Npn \@@_put_box_in_flow:
3000 {
3001   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3002   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3003   \str_if_eq:eeTF \l_@@_baseline_tl { c }
3004     { \box_use_drop:N \l_tmpa_box }
3005     \@@_put_box_in_flow_i:
3006 }

```

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

```

3007 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3008 {
3009   \pgfpicture
3010     \@@_qpoint:n { row - 1 }
3011     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3012     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3013     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3014     \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).

```

3015   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3016     {
3017       \int_set:Nn \l_tmpa_int
3018         {
3019           \str_range:Nnn
3020             \l_@@_baseline_tl
3021             6
3022             { \tl_count:o \l_@@_baseline_tl }
3023         }
3024       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3025     }
3026     {
3027       \str_if_eq:eeTF \l_@@_baseline_tl { t }
3028         { \int_set_eq:NN \l_tmpa_int \c_one_int }
3029         {
3030           \str_if_eq:onTF \l_@@_baseline_tl { b }
3031             { \int_set_eq:NN \l_tmpa_int \c@iRow }
3032             { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3033         }
3034       \bool_lazy_or:nnT
3035         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3036         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3037         {
3038           \@@_error:n { bad-value-for-baseline }
3039           \int_set_eq:NN \l_tmpa_int \c_one_int
3040         }
3041       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3042   \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }

```

```

3043     }
3044     \dim_gsub:Nn \g_tmpa_dim \pgf@y
Now, \g_tmpa_dim contains the value of the  $y$  translation we have to to.
3045     \endpgfpicture
3046     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3047     \box_use_drop:N \l_tmpa_box
3048 }

```

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).

```

3049 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3050 {

```

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.

```

3051     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3052     {
3053         \int_compare:nNnT \c@jCol > \c_one_int
3054         {
3055             \box_set_wd:Nn \l_@@_the_array_box
3056             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3057         }
3058     }

```

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

```

3059     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3060     \bool_if:NT \l_@@_caption_above_bool
3061     {
3062         \tl_if_empty:NF \l_@@_caption_tl
3063         {
3064             \bool_set_false:N \g_@@_caption_finished_bool
3065             \int_gzero:N \c@tabularnote
3066             \@@_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).

```

3067         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3068         {
3069             \tl_gput_right:Ne \g_@@_aux_tl
3070             {
3071                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3072                 { \int_use:N \g_@@_notes_caption_int }
3073             }
3074             \int_gzero:N \g_@@_notes_caption_int
3075         }
3076     }
3077 }

```

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

```

3078     \hbox
3079     {
3080         \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.

```

3081     \@@_create_extra_nodes:

```

```

3082     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3083   }

```

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

```

3084   \bool_lazy_any:nT
3085   {
3086     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3087     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3088     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3089   }
3090   \@@_insert_tabularnotes:
3091   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3092   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3093   \end { minipage }
3094 }

```

```

3095 \cs_new_protected:Npn \@@_insert_caption:
3096 {
3097   \tl_if_empty:NF \l_@@_caption_tl
3098   {
3099     \cs_if_exist:NTF \@capytype
3100     { \@@_insert_caption_i: }
3101     { \@@_error:n { caption-outside-float } }
3102   }
3103 }

```

```

3104 \cs_new_protected:Npn \@@_insert_caption_i:
3105 {
3106   \group_begin:

```

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

```

3107   \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.

```

3108   \IfPackageLoadedT { floatrow }
3109   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3110   \tl_if_empty:NTF \l_@@_short_caption_tl
3111   { \caption }
3112   { \caption [ \l_@@_short_caption_tl ] }
3113   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same `tabular` note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of `tabular` notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3114   \bool_if:NF \g_@@_caption_finished_bool
3115   {
3116     \bool_gset_true:N \g_@@_caption_finished_bool
3117     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3118     \int_gzero:N \c@tabularnote
3119   }
3120   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3121   \group_end:
3122 }

```



```

3123 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3124 {
3125   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3126   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3127 }
3128 \cs_new_protected:Npn \@@_insert_tabularnotes:
3129 {
3130   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3131   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3132   \skip_vertical:N 0.65ex

```

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

```

3133   \group_begin:
3134   \l_@@_notes_code_before_tl
3135   \tl_if_empty:NF \g_@@_tabularnote_tl
3136   {
3137     \g_@@_tabularnote_tl \par
3138     \tl_gclear:N \g_@@_tabularnote_tl
3139   }

```

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.

```

3140   \int_compare:nNnT \c@tabularnote > \c_zero_int
3141   {
3142     \bool_if:NTF \l_@@_notes_para_bool
3143     {
3144       \begin { tabularnotes* }
3145         \seq_map_inline:Nn \g_@@_notes_seq
3146         { \@@_one_tabularnote:nm ##1 }
3147         \strut
3148       \end { tabularnotes* }

```

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

```

3149     \par
3150   }
3151   {
3152     \tabularnotes
3153     \seq_map_inline:Nn \g_@@_notes_seq
3154     { \@@_one_tabularnote:nm ##1 }
3155     \strut
3156     \endtabularnotes
3157   }
3158 }
3159 \unskip
3160 \group_end:
3161 \bool_if:NT \l_@@_notes_bottomrule_bool
3162 {
3163   \IfPackageLoadedTF { booktabs }
3164   {

```

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

```

3165     \skip_vertical:N \aboverulesep

```

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

```

3166     { \CT@arc@ \hrule height \heavyrulewidth }
3167   }
3168   { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3169 }
3170 \l_@@_notes_code_after_tl
3171 \seq_gclear:N \g_@@_notes_seq
3172 \seq_gclear:N \g_@@_notes_in_caption_seq
3173 \int_gzero:N \c@tabularnote
3174 }

```

The following command will format (after the main tabular) one tabulernote (with the command `\item`). #1 is the label (when the command `\tabulernote` 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.

```

3175 \cs_set_protected:Npn \@@_one_tabulernote:nn #1
3176 {
3177   \tl_if_novalue:nTF { #1 }
3178     { \item }
3179     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3180 }

```

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`.

```

3181 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3182 {
3183   \pgfpicture
3184     \@@_qpoint:n { row - 1 }
3185     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3186     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3187     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3188   \endpgfpicture
3189   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3190   \int_if_zero:nT \l_@@_first_row_int
3191     {
3192       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3193       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3194     }
3195   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3196 }

```

Now, the general case.

```

3197 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3198 {

```

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

```

3199   \str_if_eq:eeT \l_@@_baseline_tl { t }
3200   { \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`.

```

3201   \pgfpicture
3202   \@@_qpoint:n { row - 1 }
3203   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3204   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3205     {
3206       \int_set:Nn \l_tmpa_int
3207         {
3208           \str_range:Nnn
3209             \l_@@_baseline_tl
3210             6
3211             { \tl_count:o \l_@@_baseline_tl }
3212         }
3213       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3214     }
3215   {
3216     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3217     \bool_lazy_or:nnT
3218       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3219       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3220     {
3221       \@@_error:n { bad-value-for-baseline }
3222       \int_set:Nn \l_tmpa_int 1

```

```

3223     }
3224     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3225   }
3226   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3227   \endpgfpicture
3228   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3229   \int_if_zero:nT \l_@@_first_row_int
3230   {
3231     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3232     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3233   }
3234   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3235 }

```

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.

```

3236 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3237 {

```

We will compute the real width of both delimiters used.

```

3238   \dim_zero_new:N \l_@@_real_left_delim_dim
3239   \dim_zero_new:N \l_@@_real_right_delim_dim
3240   \hbox_set:Nn \l_tmpb_box
3241   {
3242     \m@th % added 2024/11/21
3243     \c_math_toggle_token
3244     \left #1
3245     \vcenter
3246     {
3247       \vbox_to_ht:nn
3248       { \box_ht_plus_dp:N \l_tmpa_box }
3249       { }
3250     }
3251     \right .
3252     \c_math_toggle_token
3253   }
3254   \dim_set:Nn \l_@@_real_left_delim_dim
3255   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3256   \hbox_set:Nn \l_tmpb_box
3257   {
3258     \m@th % added 2024/11/21
3259     \c_math_toggle_token
3260     \left .
3261     \vbox_to_ht:nn
3262     { \box_ht_plus_dp:N \l_tmpa_box }
3263     { }
3264     \right #2
3265     \c_math_toggle_token
3266   }
3267   \dim_set:Nn \l_@@_real_right_delim_dim
3268   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3269   \skip_horizontal:N \l_@@_left_delim_dim
3270   \skip_horizontal:N -\l_@@_real_left_delim_dim
3271   \@@_put_box_in_flow:
3272   \skip_horizontal:N \l_@@_right_delim_dim
3273   \skip_horizontal:N -\l_@@_real_right_delim_dim
3274 }

```

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).

```
3275 \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, a standard error will be raised by LaTeX for incorrect nested environments).

```
3276 {
3277   \peek_remove_spaces:n
3278   {
3279     \peek_meaning:NTF \end
3280     \@@_analyze_end:Nn
3281     {
3282       \@@_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`).

```
3283       \@@_array:o \g_@@_array_preamble_tl
3284     }
3285   }
3286 }
3287 {
3288   \@@_create_col_nodes:
3289   \endarray
3290 }
```

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

```
3291 \NewDocumentEnvironment { @@-light-syntax } { b }
3292 {
```

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`.

```
3293   \tl_if_empty:nT { #1 }
3294   { \@@_fatal:n { empty-environment } }
3295   \tl_if_in:nnT { #1 } { & }
3296   { \@@_fatal:n { ampersand-in-light-syntax } }
3297   \tl_if_in:nnT { #1 } { \ }
3298   { \@@_fatal:n { double-backslash-in-light-syntax } }
```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to `no-op` before the execution of `\g_nicematrix_code_after_tl`.

```
3299   \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3300 }
```

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.

```
3301 {
3302   \@@_create_col_nodes:
3303   \endarray
3304 }
3305 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3306 {
3307   \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).

```
3308 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3309 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3310 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3311 \seq_set_split:Nee
3312 \seq_set_split:Non
3313 \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3314 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3315 \tl_if_empty:NF \l_tmpa_tl
3316 { \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.

```
3317 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3318 { \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`).

```
3319 \tl_build_begin:N \l_@@_new_body_tl
3320 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3321 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3322 \@@_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).

```
3323 \seq_map_inline:Nn \l_@@_rows_seq
3324 {
3325 \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3326 \@@_line_with_light_syntax:n { ##1 }
3327 }
3328 \tl_build_end:N \l_@@_new_body_tl
3329 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3330 {
3331 \int_set:Nn \l_@@_last_col_int
3332 { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3333 }
```

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.

```
3334 \@@_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`).

```
3335 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3336 }
3337 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3338 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3339 {
3340 \seq_clear_new:N \l_@@_cells_seq
3341 \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3342 \int_set:Nn \l_@@_nb_cols_int
3343 {
3344 \int_max:nn
3345 \l_@@_nb_cols_int
3346 { \seq_count:N \l_@@_cells_seq }
3347 }
```

```

3348 \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3349 \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3350 \seq_map_inline:Nn \l_@@_cells_seq
3351   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3352 }

```

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`.

```

3353 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3354   {
3355     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3356     { \@@_fatal:n { empty-environment } }

```

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

```

3357   \end { #2 }
3358 }

```

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`).

```

3359 \cs_new:Npn \@@_create_col_nodes:
3360   {
3361     \crrc
3362     \int_if_zero:nT \l_@@_first_col_int
3363     {
3364       \omit
3365       \hbox_overlap_left:n
3366         {
3367           \bool_if:NT \l_@@_code_before_bool
3368             { \pgfsys@markposition { \@@_env: - col - 0 } }
3369           \pgfpicture
3370           \pgfrememberpicturepositiononpagetrue
3371           \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3372           \str_if_empty:NF \l_@@_name_str
3373             { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3374           \endpgfpicture
3375           \skip_horizontal:N 2\col@sep
3376           \skip_horizontal:N \g_@@_width_first_col_dim
3377         }
3378       &
3379     }
3380     \omit

```

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

```

3381   \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`).

```

3382   \int_if_zero:nTF \l_@@_first_col_int
3383     {
3384       \bool_if:NT \l_@@_code_before_bool
3385       {
3386         \hbox
3387           {
3388             \skip_horizontal:N -0.5\arrayrulewidth
3389             \pgfsys@markposition { \@@_env: - col - 1 }
3390             \skip_horizontal:N 0.5\arrayrulewidth
3391           }
3392       }
3393       \pgfpicture
3394       \pgfrememberpicturepositiononpagetrue

```

```

3395 \pgfcoordinate { \l_@@_env: - col - 1 }
3396 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3397 \str_if_empty:NF \l_@@_name_str
3398 { \pgfnodealias { \l_@@_name_str - col - 1 } { \l_@@_env: - col - 1 } }
3399 \endpgfpicture
3400 }
3401 {
3402 \bool_if:NT \l_@@_code_before_bool
3403 {
3404 \hbox
3405 {
3406 \skip_horizontal:N 0.5\arrayrulewidth
3407 \pgfsys@markposition { \l_@@_env: - col - 1 }
3408 \skip_horizontal:N -0.5\arrayrulewidth
3409 }
3410 }
3411 \pgfpicture
3412 \pgfrememberpicturepositiononpagetrue
3413 \pgfcoordinate { \l_@@_env: - col - 1 }
3414 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3415 \str_if_empty:NF \l_@@_name_str
3416 { \pgfnodealias { \l_@@_name_str - col - 1 } { \l_@@_env: - col - 1 } }
3417 \endpgfpicture
3418 }

```

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.

```

3419 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3420 \bool_if:NF \l_@@_auto_columns_width_bool
3421 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3422 {
3423 \bool_lazy_and:nnTF
3424 \l_@@_auto_columns_width_bool
3425 { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3426 { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3427 { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3428 \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3429 }
3430 \skip_horizontal:N \g_tmpa_skip
3431 \hbox
3432 {
3433 \bool_if:NT \l_@@_code_before_bool
3434 {
3435 \hbox
3436 {
3437 \skip_horizontal:N -0.5\arrayrulewidth
3438 \pgfsys@markposition { \l_@@_env: - col - 2 }
3439 \skip_horizontal:N 0.5\arrayrulewidth
3440 }
3441 }
3442 \pgfpicture
3443 \pgfrememberpicturepositiononpagetrue
3444 \pgfcoordinate { \l_@@_env: - col - 2 }
3445 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3446 \str_if_empty:NF \l_@@_name_str
3447 { \pgfnodealias { \l_@@_name_str - col - 2 } { \l_@@_env: - col - 2 } }
3448 \endpgfpicture
3449 }

```

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.

```

3450   \int_gset_eq:NN \g_tmpa_int \c_one_int
3451   \bool_if:NTF \g_@@_last_col_found_bool
3452     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3453     { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3454     {
3455       &
3456       \omit
3457       \int_gincr:N \g_tmpa_int

```

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

```

3458       \skip_horizontal:N \g_tmpa_skip
3459       \bool_if:NT \l_@@_code_before_bool
3460       {
3461         \hbox
3462         {
3463           \skip_horizontal:N -0.5\arrayrulewidth
3464           \pgfsys@markposition
3465             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3466           \skip_horizontal:N 0.5\arrayrulewidth
3467         }
3468       }

```

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

```

3469       \pgfpicture
3470       \pgfrememberpicturepositiononpagetrue
3471       \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3472         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3473       \str_if_empty:NF \l_@@_name_str
3474         {
3475           \pgfnodealias
3476             { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3477             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3478         }
3479       \endpgfpicture
3480     }

3481     &
3482     \omit

```

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

```

3483     \int_if_zero:nT \g_@@_col_total_int
3484       { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3485     \skip_horizontal:N \g_tmpa_skip
3486     \int_gincr:N \g_tmpa_int
3487     \bool_lazy_any:nF
3488       {
3489         \g_@@_delims_bool
3490         \l_@@_tabular_bool
3491         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3492         \l_@@_exterior_arraycolsep_bool
3493         \l_@@_bar_at_end_of_pream_bool
3494       }
3495     { \skip_horizontal:N -\col@sep }
3496     \bool_if:NT \l_@@_code_before_bool
3497     {
3498       \hbox
3499       {
3500         \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.

```

3501         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3502         { \skip_horizontal:N -\arraycolsep }
3503         \pgfsys@markposition
3504         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3505         \skip_horizontal:N 0.5\arrayrulewidth
3506         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3507         { \skip_horizontal:N \arraycolsep }
3508     }
3509 }
3510 \pgfpicture
3511 \pgfrememberpicturepositiononpagetrue
3512 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3513 {
3514     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3515     {
3516         \pgfpoint
3517         { - 0.5 \arrayrulewidth - \arraycolsep }
3518         \c_zero_dim
3519     }
3520     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3521 }
3522 \str_if_empty:NF \l_@@_name_str
3523 {
3524     \pgfnodealias
3525     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3526     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3527 }
3528 \endpgfpicture

3529 \bool_if:NT \g_@@_last_col_found_bool
3530 {
3531     \hbox_overlap_right:n
3532     {
3533         \skip_horizontal:N \g_@@_width_last_col_dim
3534         \skip_horizontal:N \col@sep
3535         \bool_if:NT \l_@@_code_before_bool
3536         {
3537             \pgfsys@markposition
3538             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3539         }
3540         \pgfpicture
3541         \pgfrememberpicturepositiononpagetrue
3542         \pgfcoordinate
3543         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3544         \pgfpointorigin
3545         \str_if_empty:NF \l_@@_name_str
3546         {
3547             \pgfnodealias
3548             {
3549                 \l_@@_name_str - col
3550                 - \int_eval:n { \g_@@_col_total_int + 1 }
3551             }
3552             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3553         }
3554         \endpgfpicture
3555     }
3556 }
3557 % \cr
3558 }

```

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

```

3559 \tl_const:Nn \c_@@_preamble_first_col_tl
3560 {
3561   >
3562   {

```

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

```

3563     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3564     \bool_gset_true:N \g_@@_after_col_zero_bool
3565     \@@_begin_of_row:
3566     \hbox_set:Nw \l_@@_cell_box
3567     \@@_math_toggle:
3568     \@@_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”.

```

3569     \int_compare:nNnT \c@iRow > \c_zero_int
3570     {
3571       \bool_lazy_or:nnT
3572       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3573       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3574       {
3575         \l_@@_code_for_first_col_tl
3576         \xglobal \colorlet { nicematrix-first-col } { . }
3577       }
3578     }
3579 }

```

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`.

```

3580   l
3581   <
3582   {
3583     \@@_math_toggle:
3584     \hbox_set_end:
3585     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3586     \@@_adjust_size_box:
3587     \@@_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.

```

3588     \dim_gset:Nn \g_@@_width_first_col_dim
3589     { \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.

```

3590     \hbox_overlap_left:n
3591     {
3592       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3593       \@@_node_for_cell:
3594       { \box_use_drop:N \l_@@_cell_box }
3595       \skip_horizontal:N \l_@@_left_delim_dim
3596       \skip_horizontal:N \l_@@_left_margin_dim
3597       \skip_horizontal:N \l_@@_extra_left_margin_dim
3598     }
3599     \bool_gset_false:N \g_@@_empty_cell_bool
3600     \skip_horizontal:N -2\col@sep
3601   }
3602 }

```

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

```

3603 \tl_const:Nn \c_@@_preamble_last_col_tl
3604 {
3605   >
3606   {
3607     \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).

```
3608 \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.

```
3609 \bool_gset_true:N \g_@@_last_col_found_bool
3610 \int_gincr:N \c@jCol
3611 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3612 \hbox_set:Nw \l_@@_cell_box
3613 \@@_math_toggle:
3614 \@@_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”.

```
3615 \int_compare:nNnT \c@iRow > \c_zero_int
3616 {
3617   \bool_lazy_or:nnT
3618   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3619   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3620   {
3621     \l_@@_code_for_last_col_tl
3622     \xglobal \colorlet { nicematrix-last-col } { . }
3623   }
3624 }
3625 }
3626 l
3627 <
3628 {
3629   \@@_math_toggle:
3630   \hbox_set_end:
3631   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3632   \@@_adjust_size_box:
3633   \@@_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.

```
3634 \dim_gset:Nn \g_@@_width_last_col_dim
3635 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3636 \skip_horizontal:N -2\col@sep
```

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

```
3637 \hbox_overlap_right:n
3638 {
3639   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3640   {
3641     \skip_horizontal:N \l_@@_right_delim_dim
3642     \skip_horizontal:N \l_@@_right_margin_dim
3643     \skip_horizontal:N \l_@@_extra_right_margin_dim
3644     \@@_node_for_cell:
3645   }
3646 }
3647 \bool_gset_false:N \g_@@_empty_cell_bool
3648 }
3649 }
```

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

```
3650 \NewDocumentEnvironment { NiceArray } { }
3651 {
3652   \bool_gset_false:N \g_@@_delims_bool
3653   \str_if_empty:NT \g_@@_name_env_str
3654   { \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).

```

3655     \NiceArrayWithDelims . .
3656   }
3657   { \endNiceArrayWithDelims }

```

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

```

3658 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3659 {
3660   \NewDocumentEnvironment { #1 NiceArray } { }
3661   {
3662     \bool_gset_true:N \g_@@_delims_bool
3663     \str_if_empty:NT \g_@@_name_env_str
3664     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3665     \@@_test_if_math_mode:
3666     \NiceArrayWithDelims #2 #3
3667   }
3668   { \endNiceArrayWithDelims }
3669 }

3670 \@@_def_env:nnn p ( )
3671 \@@_def_env:nnn b [ ]
3672 \@@_def_env:nnn B \{ \}
3673 \@@_def_env:nnn v | |
3674 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3675 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3676 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3677 {
3678   \bool_set_false:N \l_@@_preamble_bool
3679   \tl_clear:N \l_tmpa_tl
3680   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3681   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3682   \tl_put_right:Nn \l_tmpa_tl
3683   {
3684     *
3685     {
3686       \int_case:nnF \l_@@_last_col_int
3687       {
3688         { -2 } { \c@MaxMatrixCols }
3689         { -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).

```

3690       }
3691       { \int_eval:n { \l_@@_last_col_int - 1 } }
3692     }
3693     { #2 }
3694   }
3695   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3696   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3697 }

3698 \clist_map_inline:nn { p , b , B , v , V }
3699 {
3700   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3701   {
3702     \bool_gset_true:N \g_@@_delims_bool

```

```

3703     \str_gset:Nn \g_@@_name_env_str { #1 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     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3711 }
3712 { \use:c { end #1 NiceArray } }
3713 }

```

We define also an environment {NiceMatrix}

```

3714 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3715 {
3716     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3717     \int_if_zero:nT \l_@@_last_col_int
3718     {
3719         \bool_set_true:N \l_@@_last_col_without_value_bool
3720         \int_set:Nn \l_@@_last_col_int { -1 }
3721     }
3722     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3723     \bool_lazy_or:nnT
3724     { \clist_if_empty_p:N \l_@@_vlines_clist }
3725     { \l_@@_except_borders_bool }
3726     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3727     \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3728 }
3729 { \endNiceArray }

```

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

```

3730 \cs_new_protected:Npn \@@_NotEmpty:
3731 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3732 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3733 {

```

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.

```

3734     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3735     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3736     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3737     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3738     \tl_if_empty:NF \l_@@_short_caption_tl
3739     {
3740         \tl_if_empty:NT \l_@@_caption_tl
3741         {
3742             \@@_error_or_warning:n { short-caption-without-caption }
3743             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3744         }
3745     }
3746     \tl_if_empty:NF \l_@@_label_tl
3747     {
3748         \tl_if_empty:NT \l_@@_caption_tl
3749         { \@@_error_or_warning:n { label-without-caption } }
3750     }
3751     \NewDocumentEnvironment { TabularNote } { b }
3752     {
3753         \bool_if:NTF \l_@@_in_code_after_bool

```

```

3754     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3755     {
3756       \tl_if_empty:NF \g_@@_tabularnote_tl
3757       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3758       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3759     }
3760   }
3761   { }
3762   \@@_settings_for_tabular:
3763   \NiceArray { #2 }
3764 }
3765 { \endNiceArray }
3766 \cs_new_protected:Npn \@@_settings_for_tabular:
3767 {
3768   \bool_set_true:N \l_@@_tabular_bool
3769   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3770   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3771   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3772 }

3773 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3774 {
3775   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3776   \dim_zero_new:N \l_@@_width_dim
3777   \dim_set:Nn \l_@@_width_dim { #1 }
3778   \keys_set:nm { nicematrix / NiceTabular } { #2 , #4 }
3779   \@@_settings_for_tabular:
3780   \NiceArray { #3 }
3781 }
3782 {
3783   \endNiceArray
3784   \int_if_zero:nT \g_@@_total_X_weight_int
3785     { \@@_error:n { NiceTabularX~without~X } }
3786 }

3787 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3788 {
3789   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3790   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3791   \keys_set:nm { nicematrix / NiceTabular } { #2 , #4 }
3792   \@@_settings_for_tabular:
3793   \NiceArray { #3 }
3794 }
3795 { \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`).

```

3796 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3797 {
3798   \bool_lazy_all:nT
3799     {
3800       { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3801       \l_@@_hvlines_bool
3802       { ! \g_@@_delims_bool }
3803       { ! \l_@@_except_borders_bool }
3804     }

```

```

3805 {
3806   \bool_set_true:N \l_@@_except_borders_bool
3807   \clist_if_empty:NF \l_@@_corners_clist
3808   { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3809   \tl_gput_right:Nn \g_@@_pre_code_after_tl
3810   {
3811     \@@_stroke_block:nnn
3812     {
3813       rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3814       draw = \l_@@_rules_color_tl
3815     }
3816     { 1-1 }
3817     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3818   }
3819 }
3820 }

```

```

3821 \cs_new_protected:Npn \@@_after_array:
3822 {

```

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`.

```

3823   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3824   \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.

```

3825   \bool_if:NT \g_@@_last_col_found_bool
3826     { \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`.

```

3827   \bool_if:NT \l_@@_last_col_without_value_bool
3828     { \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.

```

3829   \bool_if:NT \l_@@_last_row_without_value_bool
3830     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3831   \tl_gput_right:Ne \g_@@_aux_tl
3832   {
3833     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3834     {
3835       \int_use:N \l_@@_first_row_int ,
3836       \int_use:N \c@iRow ,
3837       \int_use:N \g_@@_row_total_int ,
3838       \int_use:N \l_@@_first_col_int ,
3839       \int_use:N \c@jCol ,
3840       \int_use:N \g_@@_col_total_int
3841     }
3842   }

```

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`).

```

3843   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3844   {

```

```

3845     \tl_gput_right:Ne \g_@@_aux_tl
3846     {
3847         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3848         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3849     }
3850 }
3851 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3852 {
3853     \tl_gput_right:Ne \g_@@_aux_tl
3854     {
3855         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3856         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3857         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3858         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3859     }
3860 }

```

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

```

3861     \@@_create_diag_nodes:

```

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

```

3862     \pgfpicture
3863     \int_step_inline:nn \c@iRow
3864     {
3865         \pgfnodealias
3866         { \@@_env: - ##1 - last }
3867         { \@@_env: - ##1 - \int_use:N \c@jCol }
3868     }
3869     \int_step_inline:nn \c@jCol
3870     {
3871         \pgfnodealias
3872         { \@@_env: - last - ##1 }
3873         { \@@_env: - \int_use:N \c@iRow - ##1 }
3874     }
3875     \str_if_empty:NF \l_@@_name_str
3876     {
3877         \int_step_inline:nn \c@iRow
3878         {
3879             \pgfnodealias
3880             { \l_@@_name_str - ##1 - last }
3881             { \@@_env: - ##1 - \int_use:N \c@jCol }
3882         }
3883         \int_step_inline:nn \c@jCol
3884         {
3885             \pgfnodealias
3886             { \l_@@_name_str - last - ##1 }
3887             { \@@_env: - \int_use:N \c@iRow - ##1 }
3888         }
3889     }
3890     \endpgfpicture

```

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

```

3891     \bool_if:NT \l_@@_parallelize_diags_bool
3892     {
3893         \int_gzero_new:N \g_@@_ddots_int
3894         \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`

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

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.

```

3895     \dim_gzero_new:N \g_@@_delta_x_one_dim
3896     \dim_gzero_new:N \g_@@_delta_y_one_dim
3897     \dim_gzero_new:N \g_@@_delta_x_two_dim
3898     \dim_gzero_new:N \g_@@_delta_y_two_dim
3899   }

3900   \int_zero_new:N \l_@@_initial_i_int
3901   \int_zero_new:N \l_@@_initial_j_int
3902   \int_zero_new:N \l_@@_final_i_int
3903   \int_zero_new:N \l_@@_final_j_int
3904   \bool_set_false:N \l_@@_initial_open_bool
3905   \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.

```

3906   \bool_if:NT \l_@@_small_bool
3907   {
3908     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3909     \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }

```

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

```

3910     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3911       { 0.6 \l_@@_xdots_shorten_start_dim }
3912     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3913       { 0.6 \l_@@_xdots_shorten_end_dim }
3914   }

```

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

```

3915   \@@_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.

```

3916   \clist_if_empty:NF \l_@@_corners_clist
3917   {
3918     \bool_if:NTF \l_@@_no_cell_nodes_bool
3919       { \@@_error:n { corners~with~no~cell~nodes } }
3920       { \@@_compute_corners: }
3921   }

```

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

```

3922   \@@_adjust_pos_of_blocks_seq:
3923   \@@_deal_with_rounded_corners:
3924   \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3925   \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3926   \IfPackageLoadedT { tikz }
3927   {
3928     \tikzset
3929     {
3930       every-picture / .style =
3931       {
3932         overlay ,
3933         remember-picture ,
3934         name-prefix = \@@_env: -

```

```

3935     }
3936   }
3937 }
3938 \bool_if:NT \c_@@_recent_array_bool
3939   { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3940 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3941 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3942 \cs_set_eq:NN \OverBrace \@@_OverBrace
3943 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3944 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3945 \cs_set_eq:NN \line \@@_line

```

The LaTeX-style boolean `\ifmeasuring@` is used by `amsmath` during the phase of measure in environments such as `{align}`, etc.

```

3946 \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
3947 \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.

```

3948 \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).

```

3949 \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`).

```

3950 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
3951   { \@@_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:`.

```

3952 \bool_set_true:N \l_@@_in_code_after_bool
3953 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3954 \scan_stop:
3955 \tl_gclear:N \g_nicematrix_code_after_tl
3956 \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.

```

3957 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3958 \tl_if_empty:NF \g_@@_pre_code_before_tl
3959   {
3960     \tl_gput_right:Ne \g_@@_aux_tl
3961     {
3962       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3963         { \exp_not:o \g_@@_pre_code_before_tl }
3964     }
3965     \tl_gclear:N \g_@@_pre_code_before_tl
3966   }
3967 \tl_if_empty:NF \g_nicematrix_code_before_tl
3968   {
3969     \tl_gput_right:Ne \g_@@_aux_tl
3970     {
3971       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3972         { \exp_not:o \g_nicematrix_code_before_tl }
3973     }
3974     \tl_gclear:N \g_nicematrix_code_before_tl
3975   }

```

```

3976 \str_gclear:N \g_@@_name_env_str
3977 \@@_restore_iRow_jCol:

```

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

```

3978 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3979 }

```

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`.

```

3980 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3981 { \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).

```

3982 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3983 {
3984 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3985 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3986 }

```

The following command must *not* be protected.

```

3987 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3988 {
3989 { #1 }
3990 { #2 }
3991 {
3992 \int_compare:nNnTF { #3 } > { 98 }
3993 { \int_use:N \c@iRow }
3994 { #3 }
3995 }
3996 {
3997 \int_compare:nNnTF { #4 } > { 98 }
3998 { \int_use:N \c@jCol }
3999 { #4 }
4000 }
4001 { #5 }
4002 }

```

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).

```

4003 \hook_gput_code:nnn { begindocument } { . }
4004 {
4005 \cs_new_protected:Npe \@@_draw_dotted_lines:
4006 {
4007 \c_@@_pgfortikzpicture_tl
4008 \@@_draw_dotted_lines_i:
4009 \c_@@_endpgfortikzpicture_tl
4010 }
4011 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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

```

4012 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4013 {
4014   \pgfrememberpicturepositiononpagetrue
4015   \pgf@relevantforpicturesizefalse
4016   \g_@@_HVdotsfor_lines_tl
4017   \g_@@_Vdots_lines_tl
4018   \g_@@_Ddots_lines_tl
4019   \g_@@_Iddots_lines_tl
4020   \g_@@_Cdots_lines_tl
4021   \g_@@_Ldots_lines_tl
4022 }

4023 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4024 {
4025   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4026   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4027 }

```

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

```

4028 \pgfdeclareshape { @@_diag_node }
4029 {
4030   \savedanchor { \five }
4031   {
4032     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4033     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4034   }
4035   \anchor { 5 } { \five }
4036   \anchor { center } { \pgfpointorigin }
4037   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4038   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4039   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4040   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4041   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4042   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4043   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4044   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4045   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4046   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4047 }

```

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).

```

4048 \cs_new_protected:Npn \@@_create_diag_nodes:
4049 {
4050   \pgfpicture
4051   \pgfrememberpicturepositiononpagetrue
4052   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4053   {
4054     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4055     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4056     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4057     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4058     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4059     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4060     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4061     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4062     \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.

```

4063     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4064     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4065     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4066     \str_if_empty:NF \l_@@_name_str
4067     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4068   }

```

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

```

4069     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4070     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4071     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4072     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4073     \pgfcoordinate
4074     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4075     \pgfnodealias
4076     { \@@_env: - last }
4077     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4078     \str_if_empty:NF \l_@@_name_str
4079     {
4080       \pgfnodealias
4081       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4082       { \@@_env: - \int_use:N \l_tmpa_int }
4083       \pgfnodealias
4084       { \l_@@_name_str - last }
4085       { \@@_env: - last }
4086     }
4087     \endpgfpicture
4088   }

```

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.

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

The command `\@@_find_extremities_of_line:nmmn` 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.

```
4089 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4090 {
```

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

```
4091 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4092 \int_set:Nn \l_@@_initial_i_int { #1 }
4093 \int_set:Nn \l_@@_initial_j_int { #2 }
4094 \int_set:Nn \l_@@_final_i_int { #1 }
4095 \int_set:Nn \l_@@_final_j_int { #2 }
```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```
4096 \bool_set_false:N \l_@@_stop_loop_bool
4097 \bool_do_until:Nn \l_@@_stop_loop_bool
4098 {
4099 \int_add:Nn \l_@@_final_i_int { #3 }
4100 \int_add:Nn \l_@@_final_j_int { #4 }
4101 \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.

```
4102 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4103 \if_int_compare:w #3 = \c_one_int
4104 \bool_set_true:N \l_@@_final_open_bool
4105 \else:
4106 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4107 \bool_set_true:N \l_@@_final_open_bool
4108 \fi:
4109 \fi:
4110 \else:
4111 \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4112 \if_int_compare:w #4 = -1
4113 \bool_set_true:N \l_@@_final_open_bool
4114 \fi:
4115 \else:
4116 \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4117 \if_int_compare:w #4 = \c_one_int
4118 \bool_set_true:N \l_@@_final_open_bool
4119 \fi:
4120 \fi:
4121 \fi:
4122 \fi:
4123 \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.

```
4124 {
```

We do a step backwards.

```
4125 \int_sub:Nn \l_@@_final_i_int { #3 }
4126 \int_sub:Nn \l_@@_final_j_int { #4 }
4127 \bool_set_true:N \l_@@_stop_loop_bool
4128 }
```

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`.

```
4129 {
4130 \cs_if_exist:cTF
4131 {
4132 @@ _ dotted _
```

```

4133         \int_use:N \l_@@_final_i_int -
4134         \int_use:N \l_@@_final_j_int
4135     }
4136     {
4137         \int_sub:Nn \l_@@_final_i_int { #3 }
4138         \int_sub:Nn \l_@@_final_j_int { #4 }
4139         \bool_set_true:N \l_@@_final_open_bool
4140         \bool_set_true:N \l_@@_stop_loop_bool
4141     }
4142     {
4143         \cs_if_exist:cTF
4144         {
4145             pgf @ sh @ ns @ \@@_env:
4146             - \int_use:N \l_@@_final_i_int
4147             - \int_use:N \l_@@_final_j_int
4148         }
4149         { \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.

```

4150         {
4151             \cs_set_nopar:cpn
4152             {
4153                 @@ _ dotted _
4154                 \int_use:N \l_@@_final_i_int -
4155                 \int_use:N \l_@@_final_j_int
4156             }
4157             { }
4158         }
4159     }
4160 }
4161 }

```

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

```

4162     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4163     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4164     \bool_do_until:Nn \l_@@_stop_loop_bool
4165     {
4166         \int_sub:Nn \l_@@_initial_i_int { #3 }
4167         \int_sub:Nn \l_@@_initial_j_int { #4 }
4168         \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.

```

4169         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4170         \if_int_compare:w #3 = \c_one_int
4171         \bool_set_true:N \l_@@_initial_open_bool
4172     \else:

```

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

```

4173         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4174         \bool_set_true:N \l_@@_initial_open_bool
4175     \fi:
4176     \fi:
4177 \else:
4178     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4179     \if_int_compare:w #4 = \c_one_int

```

```

4180         \bool_set_true:N \l_@@_initial_open_bool
4181     \fi:
4182 \else:
4183     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4184         \if_int_compare:w #4 = -1
4185             \bool_set_true:N \l_@@_initial_open_bool
4186         \fi:
4187     \fi:
4188 \fi:
4189 \fi:
4190 \bool_if:NTF \l_@@_initial_open_bool
4191 {
4192     \int_add:Nn \l_@@_initial_i_int { #3 }
4193     \int_add:Nn \l_@@_initial_j_int { #4 }
4194     \bool_set_true:N \l_@@_stop_loop_bool
4195 }
4196 {
4197     \cs_if_exist:cTF
4198     {
4199         @@ _ dotted _
4200         \int_use:N \l_@@_initial_i_int -
4201         \int_use:N \l_@@_initial_j_int
4202     }
4203     {
4204         \int_add:Nn \l_@@_initial_i_int { #3 }
4205         \int_add:Nn \l_@@_initial_j_int { #4 }
4206         \bool_set_true:N \l_@@_initial_open_bool
4207         \bool_set_true:N \l_@@_stop_loop_bool
4208     }
4209     {
4210         \cs_if_exist:cTF
4211         {
4212             pgf @ sh @ ns @ \@@_env:
4213             - \int_use:N \l_@@_initial_i_int
4214             - \int_use:N \l_@@_initial_j_int
4215         }
4216         { \bool_set_true:N \l_@@_stop_loop_bool }
4217         {
4218             \cs_set_nopar:cpn
4219             {
4220                 @@ _ dotted _
4221                 \int_use:N \l_@@_initial_i_int -
4222                 \int_use:N \l_@@_initial_j_int
4223             }
4224             { }
4225         }
4226     }
4227 }
4228 }

```

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.

```

4229     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4230     {
4231         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4232         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4233         { \int_use:N \l_@@_final_i_int }
4234         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4235         { } % for the name of the block
4236     }
4237 }

```


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

```

4238 \cs_new_protected:Npn \@@_open_shorten:
4239 {
4240   \bool_if:NT \l_@@_initial_open_bool
4241     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4242   \bool_if:NT \l_@@_final_open_bool
4243     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4244 }

```

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

```

4245 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4246 {
4247   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4248   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4249   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4250   \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`.

```

4251   \seq_if_empty:NF \g_@@_submatrix_seq
4252     {
4253       \seq_map_inline:Nn \g_@@_submatrix_seq
4254         { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4255     }
4256 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix we are analyzing.

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

```

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

```

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

```

4257 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4258 {
4259   \if_int_compare:w #3 > #1
4260     \else:
4261       \if_int_compare:w #1 > #5

```

```

4262 \else:
4263 \if_int_compare:w #4 > #2
4264 \else:
4265 \if_int_compare:w #2 > #6
4266 \else:
4267 \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4268 \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4269 \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4270 \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4271 \fi:
4272 \fi:
4273 \fi:
4274 \fi:
4275 }

4276 \cs_new_protected:Npn \@@_set_initial_coords:
4277 {
4278 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4279 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4280 }
4281 \cs_new_protected:Npn \@@_set_final_coords:
4282 {
4283 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4284 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4285 }
4286 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4287 {
4288 \pgfpointanchor
4289 {
4290 \@@_env:
4291 - \int_use:N \l_@@_initial_i_int
4292 - \int_use:N \l_@@_initial_j_int
4293 }
4294 { #1 }
4295 \@@_set_initial_coords:
4296 }
4297 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4298 {
4299 \pgfpointanchor
4300 {
4301 \@@_env:
4302 - \int_use:N \l_@@_final_i_int
4303 - \int_use:N \l_@@_final_j_int
4304 }
4305 { #1 }
4306 \@@_set_final_coords:
4307 }
4308 \cs_new_protected:Npn \@@_open_x_initial_dim:
4309 {
4310 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4311 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4312 {
4313 \cs_if_exist:cT
4314 { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4315 {
4316 \pgfpointanchor
4317 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4318 { west }
4319 \dim_set:Nn \l_@@_x_initial_dim
4320 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4321 }
4322 }

```

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

```

4323 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4324 {
4325   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4326   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4327   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4328 }
4329 }

4330 \cs_new_protected:Npn \@@_open_x_final_dim:
4331 {
4332   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4333   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4334   {
4335     \cs_if_exist:cT
4336     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4337     {
4338       \pgfpointanchor
4339       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4340       { east }
4341       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4342       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4343     }
4344 }

```

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

```

4345 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4346 {
4347   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4348   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4349   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4350 }
4351 }

```

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.

```

4352 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4353 {
4354   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4355   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4356   {
4357     \@@_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.

```

4358   \group_begin:
4359   \@@_open_shorten:
4360   \int_if_zero:nTF { #1 }
4361   { \color { nicematrix-first-row } }
4362   {

```

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.

```

4363     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4364     { \color { nicematrix-last-row } }
4365   }
4366   \keys_set:nn { nicematrix / xdots } { #3 }
4367   \@@_color:o \l_@@_xdots_color_tl
4368   \@@_actually_draw_Ldots:
4369   \group_end:
4370 }
4371 }

```

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`.

```

4372 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4373 {
4374   \bool_if:NTF \l_@@_initial_open_bool
4375   {
4376     \@@_open_x_initial_dim:
4377     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4378     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4379   }
4380   { \@@_set_initial_coords_from_anchor:n { base-east } }
4381 \bool_if:NTF \l_@@_final_open_bool
4382 {
4383   \@@_open_x_final_dim:
4384   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4385   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4386 }
4387 { \@@_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.

```

4388 \bool_lazy_all:nTF
4389 {
4390   \l_@@_initial_open_bool
4391   \l_@@_final_open_bool
4392   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4393 }
4394 {
4395   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4396   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4397 }

```

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 (?).

```

4398 {
4399   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4400   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4401 }
4402 \@@_draw_line:
4403 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

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

```

4410     \group_begin:
4411     \@@_open_shorten:
4412     \int_if_zero:nTF { #1 }
4413     { \color { nicematrix-first-row } }
4414     {

```

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.

```

4415         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4416         { \color { nicematrix-last-row } }
4417     }
4418     \keys_set:nn { nicematrix / xdots } { #3 }
4419     \@@_color:o \l_@@_xdots_color_tl
4420     \@@_actually_draw_Cdots:
4421 \group_end:
4422 }
4423 }

```

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`.

```

4424 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4425 {
4426     \bool_if:NTF \l_@@_initial_open_bool
4427     { \@@_open_x_initial_dim: }
4428     { \@@_set_initial_coords_from_anchor:n { mid-east } } }
4429 \bool_if:NTF \l_@@_final_open_bool
4430 { \@@_open_x_final_dim: }
4431 { \@@_set_final_coords_from_anchor:n { mid-west } } }
4432 \bool_lazy_and:nnTF
4433 \l_@@_initial_open_bool
4434 \l_@@_final_open_bool
4435 {
4436     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4437     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4438     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } } }
4439 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4440 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4441 }
4442 {
4443     \bool_if:NT \l_@@_initial_open_bool
4444     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4445     \bool_if:NT \l_@@_final_open_bool
4446     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4447 } }
4448 \@@_draw_line:
4449 }
4450 \cs_new_protected:Npn \@@_open_y_initial_dim:
4451 {
4452     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4453     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4454     {

```

```

4455 \cs_if_exist:cT
4456 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4457 {
4458 \pgfpointanchor
4459 { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4460 { north }
4461 \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4462 { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4463 }
4464 }
4465 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4466 {
4467 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4468 \dim_set:Nn \l_@@_y_initial_dim
4469 {
4470 \fp_to_dim:n
4471 {
4472 \pgf@y
4473 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4474 }
4475 }
4476 }
4477 }
4478 \cs_new_protected:Npn \@@_open_y_final_dim:
4479 {
4480 \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4481 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4482 {
4483 \cs_if_exist:cT
4484 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4485 {
4486 \pgfpointanchor
4487 { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4488 { south }
4489 \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4490 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4491 }
4492 }
4493 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4494 {
4495 \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4496 \dim_set:Nn \l_@@_y_final_dim
4497 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4498 }
4499 }

```

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.

```

4500 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4501 {
4502 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4503 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4504 {
4505 \@@_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.

```

4506 \group_begin:
4507 \@@_open_shorten:
4508 \int_if_zero:nTF { #2 }
4509 { \color { nicematrix-first-col } }
4510 {
4511 \int_compare:nNnT { #2 } = \l_@@_last_col_int
4512 { \color { nicematrix-last-col } }

```

```

4513     }
4514     \keys_set:nn { nicematrix / xdots } { #3 }
4515     \@@_color:o \l_@@_xdots_color_tl
4516     \@@_actually_draw_Vdots:
4517     \group_end:
4518   }
4519 }

```

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`.

```

4520 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4521 {

```

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

```

4522   \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.

```

4523   {
4524     \@@_open_y_initial_dim:
4525     \@@_open_y_final_dim:
4526     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4527     {
4528       \@@_qpoint:n { col - 1 }
4529       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4530       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4531       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4532       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4533     }
4534     {
4535       \bool_lazy_and:nnTF
4536         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4537         { \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”.

```

4538     {
4539       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4540       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4541       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4542       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4543       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4544     }

```

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

```

4545     {
4546       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4547       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4548       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4549       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4550     }
4551   }
4552 }

```

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.

```

4553     {
4554         \bool_set_false:N \l_tmpa_bool
4555         \bool_if:NF \l_@@_initial_open_bool
4556         {
4557             \bool_if:NF \l_@@_final_open_bool
4558             {
4559                 \@@_set_initial_coords_from_anchor:n { south-west }
4560                 \@@_set_final_coords_from_anchor:n { north-west }
4561                 \bool_set:Nn \l_tmpa_bool
4562                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4563             }
4564         }

```

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

```

4565         \bool_if:NTF \l_@@_initial_open_bool
4566         {
4567             \@@_open_y_initial_dim:
4568             \@@_set_final_coords_from_anchor:n { north }
4569             \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4570         }
4571         {
4572             \@@_set_initial_coords_from_anchor:n { south }
4573             \bool_if:NTF \l_@@_final_open_bool
4574             \@@_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.

```

4575         {
4576             \@@_set_final_coords_from_anchor:n { north }
4577             \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4578             {
4579                 \dim_set:Nn \l_@@_x_initial_dim
4580                 {
4581                     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4582                     \l_@@_x_initial_dim \l_@@_x_final_dim
4583                 }
4584             }
4585         }
4586     }
4587 }
4588 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4589 \@@_draw_line:
4590 }

```

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.

```

4591 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4592 {
4593     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4594     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4595     {
4596         \@@_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.

```

4597         \group_begin:
4598         \@@_open_shorten:

```



```

4599     \keys_set:nn { nicematrix / xdots } { #3 }
4600     \@@_color:o \l_@@_xdots_color_tl
4601     \@@_actually_draw_Ddots:
4602     \group_end:
4603   }
4604 }

```

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.`

```

4605 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4606 {
4607   \bool_if:NTF \l_@@_initial_open_bool
4608   {
4609     \@@_open_y_initial_dim:
4610     \@@_open_x_initial_dim:
4611   }
4612   { \@@_set_initial_coords_from_anchor:n { south-east } }
4613   \bool_if:NTF \l_@@_final_open_bool
4614   {
4615     \@@_open_x_final_dim:
4616     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4617   }
4618   { \@@_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.

```

4619   \bool_if:NT \l_@@_parallelize_diags_bool
4620   {
4621     \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).

```

4622     \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.

```

4623     {
4624       \dim_gset:Nn \g_@@_delta_x_one_dim
4625       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4626       \dim_gset:Nn \g_@@_delta_y_one_dim
4627       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4628     }

```

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`.

```

4629     {
4630       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4631       {
4632         \dim_set:Nn \l_@@_y_final_dim
4633         {
4634           \l_@@_y_initial_dim +
4635           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4636           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4637         }

```

```

4638     }
4639   }
4640 }
4641 \@@_draw_line:
4642 }

```

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.

```

4643 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4644 {
4645   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4646   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4647   {
4648     \@@_find_extremities_of_line:nmmm { #1 } { #2 } 1 { -1 }

```

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

```

4649     \group_begin:
4650     \@@_open_shorten:
4651     \keys_set:nn { nicematrix / xdots } { #3 }
4652     \@@_color:o \l_@@_xdots_color_tl
4653     \@@_actually_draw_Iddots:
4654   \group_end:
4655 }
4656 }

```

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`.

```

4657 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4658 {
4659   \bool_if:NTF \l_@@_initial_open_bool
4660   {
4661     \@@_open_y_initial_dim:
4662     \@@_open_x_initial_dim:
4663   }
4664   { \@@_set_initial_coords_from_anchor:n { south-west } }
4665   \bool_if:NTF \l_@@_final_open_bool
4666   {
4667     \@@_open_y_final_dim:
4668     \@@_open_x_final_dim:
4669   }
4670   { \@@_set_final_coords_from_anchor:n { north-east } }
4671   \bool_if:NT \l_@@_parallelize_diags_bool
4672   {
4673     \int_gincr:N \g_@@_iddots_int
4674     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4675     {
4676       \dim_gset:Nn \g_@@_delta_x_two_dim
4677       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4678       \dim_gset:Nn \g_@@_delta_y_two_dim
4679       { \l_@@_y_final_dim - \l_@@_y_initial_dim }

```

```

4680     }
4681     {
4682         \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4683         {
4684             \dim_set:Nn \l_@@_y_final_dim
4685             {
4686                 \l_@@_y_initial_dim +
4687                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4688                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4689             }
4690         }
4691     }
4692 }
4693 \@@_draw_line:
4694 }

```

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`

```

4695 \cs_new_protected:Npn \@@_draw_line:
4696 {
4697     \pgfrememberpicturepositiononpagetrue
4698     \pgf@relevantforpicturesizefalse
4699     \bool_lazy_or:nnTF
4700     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4701     \l_@@_dotted_bool
4702     \@@_draw_standard_dotted_line:
4703     \@@_draw_unstandard_dotted_line:
4704 }

```

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.

```

4705 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4706 {
4707     \begin { scope }
4708     \@@_draw_unstandard_dotted_line:o
4709     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4710 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

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

```

4711 \generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4712 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4713 {

```

```

4714 \l_@@_draw_unstandard_dotted_line:nnoo
4715 { #1 }
4716 \l_@@_xdots_up_tl
4717 \l_@@_xdots_down_tl
4718 \l_@@_xdots_middle_tl
4719 }

```

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

```

4720 \hook_gput_code:nnn { begindocument } { . }
4721 {
4722   \IfPackageLoadedT { tikz }
4723   {
4724     \tikzset
4725     {
4726       @@_node_above / .style = { sloped , above } ,
4727       @@_node_below / .style = { sloped , below } ,
4728       @@_node_middle / .style =
4729       {
4730         sloped ,
4731         inner~sep = \c_@@_innersep_middle_dim
4732       }
4733     }
4734   }
4735 }

4736 \cs_generate_variant:Nn \l_@@_draw_unstandard_dotted_line:nnnn { n o o o }
4737 \cs_new_protected:Npn \l_@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4738 {

```

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.

```

4739 \dim_zero_new:N \l_@@_l_dim
4740 \dim_set:Nn \l_@@_l_dim
4741 {
4742   \fp_to_dim:n
4743   {
4744     sqrt
4745     (
4746       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4747       +
4748       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4749     )
4750   }
4751 }

```

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.

```

4752 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4753 {
4754   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4755   \l_@@_draw_unstandard_dotted_line_i:
4756 }

```

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

```

4757 \bool_if:NT \l_@@_xdots_h_labels_bool
4758 {

```

```

4759     \tikzset
4760     {
4761         @@_node_above / .style = { auto = left } ,
4762         @@_node_below / .style = { auto = right } ,
4763         @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4764     }
4765 }
4766 \tl_if_empty:nF { #4 }
4767 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4768 \draw
4769 [ #1 ]
4770 ( \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).

```

4771     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4772     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4773     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4774     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4775 \end { scope }
4776 }
4777 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4778 {
4779     \dim_set:Nn \l_tmpa_dim
4780     {
4781         \l_@@_x_initial_dim
4782         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4783         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4784     }
4785     \dim_set:Nn \l_tmpb_dim
4786     {
4787         \l_@@_y_initial_dim
4788         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4789         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4790     }
4791     \dim_set:Nn \l_@@_tmpc_dim
4792     {
4793         \l_@@_x_final_dim
4794         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4795         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4796     }
4797     \dim_set:Nn \l_@@_tmpd_dim
4798     {
4799         \l_@@_y_final_dim
4800         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4801         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4802     }
4803     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4804     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4805     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4806     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4807 }

```

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

```

4808 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4809 {
4810     \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.

```

4811     \dim_zero_new:N \l_@@_l_dim
4812     \dim_set:Nn \l_@@_l_dim

```

```

4813     {
4814         \fp_to_dim:n
4815         {
4816             sqrt
4817             (
4818                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4819                 +
4820                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4821             )
4822         }
4823     }

```

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.

```

4824     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4825     {
4826         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4827         \@@_draw_standard_dotted_line_i:
4828     }
4829 \group_end:
4830 \bool_lazy_all:nF
4831 {
4832     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4833     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4834     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4835 }
4836 \l_@@_labels_standard_dotted_line:
4837 }
4838 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4839 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4840 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4841     \int_set:Nn \l_tmpa_int
4842     {
4843         \dim_ratio:nn
4844         {
4845             \l_@@_l_dim
4846             - \l_@@_xdots_shorten_start_dim
4847             - \l_@@_xdots_shorten_end_dim
4848         }
4849         \l_@@_xdots_inter_dim
4850     }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4851     \dim_set:Nn \l_tmpa_dim
4852     {
4853         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4854         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4855     }
4856     \dim_set:Nn \l_tmpb_dim
4857     {
4858         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4859         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4860     }

```

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.

```

4861     \dim_gadd:Nn \l_@@_x_initial_dim

```

```

4862 {
4863   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4864   \dim_ratio:nn
4865   {
4866     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4867     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4868   }
4869   { 2 \l_@@_l_dim }
4870 }
4871 \dim_gadd:Nn \l_@@_y_initial_dim
4872 {
4873   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4874   \dim_ratio:nn
4875   {
4876     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4877     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4878   }
4879   { 2 \l_@@_l_dim }
4880 }
4881 \pgf@relevantforpicturesizefalse
4882 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4883 {
4884   \pgfpathcircle
4885   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4886   { \l_@@_xdots_radius_dim }
4887   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4888   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4889 }
4890 \pgfusepathqfill
4891 }

4892 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4893 {
4894   \pgfscope
4895   \pgftransformshift
4896   {
4897     \pgfpointlineattime { 0.5 }
4898     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4899     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4900   }
4901   \fp_set:Nn \l_tmpa_fp
4902   {
4903     atand
4904     (
4905       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4906       \l_@@_x_final_dim - \l_@@_x_initial_dim
4907     )
4908   }
4909   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4910   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4911   \tl_if_empty:NF \l_@@_xdots_middle_tl
4912   {
4913     \begin { pgfscope }
4914     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4915     \pgfnode
4916     { rectangle }
4917     { center }
4918     {
4919       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4920       {
4921         \c_math_toggle_token
4922         \scriptstyle \l_@@_xdots_middle_tl
4923         \c_math_toggle_token

```

```

4924     }
4925   }
4926   { }
4927   {
4928     \pgfsetfillcolor { white }
4929     \pgfusepath { fill }
4930   }
4931   \end { pgfscope }
4932 }
4933 \tl_if_empty:NF \l_@@_xdots_up_tl
4934 {
4935   \pgfnode
4936   { rectangle }
4937   { south }
4938   {
4939     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4940     {
4941       \c_math_toggle_token
4942       \scriptstyle \l_@@_xdots_up_tl
4943       \c_math_toggle_token
4944     }
4945   }
4946   { }
4947   { \pgfusepath { } }
4948 }
4949 \tl_if_empty:NF \l_@@_xdots_down_tl
4950 {
4951   \pgfnode
4952   { rectangle }
4953   { north }
4954   {
4955     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4956     {
4957       \c_math_toggle_token
4958       \scriptstyle \l_@@_xdots_down_tl
4959       \c_math_toggle_token
4960     }
4961   }
4962   { }
4963   { \pgfusepath { } }
4964 }
4965 \endpgfscope
4966 }

```

18 User commands available in the new environments

The commands `\@@_Ldots`, `\@@_Cdots`, `\@@_Vdots`, `\@@_Ddots` and `\@@_Iddots` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of `nicematrix` rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

4967 \hook_gput_code:nnn { begindocument } { . }
4968 {
4969   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4970   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```



```

4971 \cs_new_protected:Npn \@@_Ldots
4972   { \@@_collect_options:n { \@@_Ldots_i } }
4973 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4974   {
4975     \int_if_zero:nTF \c@jCol
4976       { \@@_error:nn { in~first~col } \Ldots }
4977     {
4978       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4979         { \@@_error:nn { in~last~col } \Ldots }
4980       {
4981         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4982         { #1 , down = #2 , up = #3 , middle = #4 }
4983       }
4984     }
4985     \bool_if:NF \l_@@_nullify_dots_bool
4986     { \phantom { \ensuremath { \@@_old_ldots } } }
4987     \bool_gset_true:N \g_@@_empty_cell_bool
4988   }

4989 \cs_new_protected:Npn \@@_Cdots
4990   { \@@_collect_options:n { \@@_Cdots_i } }
4991 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4992   {
4993     \int_if_zero:nTF \c@jCol
4994       { \@@_error:nn { in~first~col } \Cdots }
4995     {
4996       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4997         { \@@_error:nn { in~last~col } \Cdots }
4998       {
4999         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5000         { #1 , down = #2 , up = #3 , middle = #4 }
5001       }
5002     }
5003     \bool_if:NF \l_@@_nullify_dots_bool
5004     { \phantom { \ensuremath { \@@_old_cdots } } }
5005     \bool_gset_true:N \g_@@_empty_cell_bool
5006   }

5007 \cs_new_protected:Npn \@@_Vdots
5008   { \@@_collect_options:n { \@@_Vdots_i } }
5009 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5010   {
5011     \int_if_zero:nTF \c@iRow
5012       { \@@_error:nn { in~first~row } \Vdots }
5013     {
5014       \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5015         { \@@_error:nn { in~last~row } \Vdots }
5016       {
5017         \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5018         { #1 , down = #2 , up = #3 , middle = #4 }
5019       }
5020     }
5021     \bool_if:NF \l_@@_nullify_dots_bool
5022     { \phantom { \ensuremath { \@@_old_vdots } } }
5023     \bool_gset_true:N \g_@@_empty_cell_bool
5024   }

5025 \cs_new_protected:Npn \@@_Ddots
5026   { \@@_collect_options:n { \@@_Ddots_i } }
5027 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5028   {

```

```

5029 \int_case:nnF \c@iRow
5030 {
5031     0          { \@@_error:nn { in-first-row } \Ddots }
5032     \l_@@_last_row_int { \@@_error:nn { in-last-row } \Ddots }
5033 }
5034 {
5035     \int_case:nnF \c@jCol
5036     {
5037         0          { \@@_error:nn { in-first-col } \Ddots }
5038         \l_@@_last_col_int { \@@_error:nn { in-last-col } \Ddots }
5039     }
5040     {
5041         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5042         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5043         { #1 , down = #2 , up = #3 , middle = #4 }
5044     }
5045 }
5046 }
5047 \bool_if:NF \l_@@_nullify_dots_bool
5048 { \phantom { \ensuremath { \@@_old_ddots } } }
5049 \bool_gset_true:N \g_@@_empty_cell_bool
5050 }

5051 \cs_new_protected:Npn \@@_Iddots
5052 { \@@_collect_options:n { \@@_Iddots_i } }
5053 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5054 {
5055     \int_case:nnF \c@iRow
5056     {
5057         0          { \@@_error:nn { in-first-row } \Iddots }
5058         \l_@@_last_row_int { \@@_error:nn { in-last-row } \Iddots }
5059     }
5060     {
5061         \int_case:nnF \c@jCol
5062         {
5063             0          { \@@_error:nn { in-first-col } \Iddots }
5064             \l_@@_last_col_int { \@@_error:nn { in-last-col } \Iddots }
5065         }
5066         {
5067             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5068             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5069             { #1 , down = #2 , up = #3 , middle = #4 }
5070         }
5071     }
5072     \bool_if:NF \l_@@_nullify_dots_bool
5073     { \phantom { \ensuremath { \@@_old_iddots } } }
5074     \bool_gset_true:N \g_@@_empty_cell_bool
5075 }
5076 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5077 \keys_define:nn { nicematrix / Ddots }
5078 {
5079     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5080     draw-first .default:n = true ,
5081     draw-first .value_forbidden:n = true
5082 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5083 \cs_new_protected:Npn \@@_Hspace:

```

```

5084 {
5085   \bool_gset_true:N \g_@@_empty_cell_bool
5086   \hspace
5087 }

```

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`.

```

5088 \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`.

```

5089 \cs_new:Npn \@@_Hdotsfor:
5090 {
5091   \bool_lazy_and:nnTF
5092     { \int_if_zero_p:n \c@jCol }
5093     { \int_if_zero_p:n \l_@@_first_col_int }
5094     {
5095       \bool_if:NTF \g_@@_after_col_zero_bool
5096       {
5097         \multicolumn { 1 } { c } { }
5098         \@@_Hdotsfor_i
5099       }
5100       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5101     }
5102   {
5103     \multicolumn { 1 } { c } { }
5104     \@@_Hdotsfor_i
5105   }
5106 }

```

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:`).

```

5107 \hook_gput_code:nnn { begindocument } { . }
5108 {
5109   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5110   \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.

```

5111   \cs_new_protected:Npn \@@_Hdotsfor_i
5112     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5113   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5114     {
5115       \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5116       {
5117         \@@_Hdotsfor:nnnn
5118         { \int_use:N \c@iRow }
5119         { \int_use:N \c@jCol }
5120         { #2 }
5121         {
5122           #1 , #3 ,
5123           down = \exp_not:n { #4 } ,
5124           up = \exp_not:n { #5 } ,
5125           middle = \exp_not:n { #6 }
5126         }
5127       }
5128     \prg_replicate:nn { #2 - 1 }
5129     {
5130       &
5131       \multicolumn { 1 } { c } { }

```

```

5132         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5133     }
5134 }
5135 }

```

```

5136 \cs_new_protected:Npn \@@_Hdotsfor:nmmm #1 #2 #3 #4
5137 {
5138     \bool_set_false:N \l_@@_initial_open_bool
5139     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5140     \int_set:Nn \l_@@_initial_i_int { #1 }
5141     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5142     \int_compare:nNnTF { #2 } = \c_one_int
5143     {
5144         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5145         \bool_set_true:N \l_@@_initial_open_bool
5146     }
5147     {
5148         \cs_if_exist:cTF
5149         {
5150             pgf @ sh @ ns @ \@@_env:
5151             - \int_use:N \l_@@_initial_i_int
5152             - \int_eval:n { #2 - 1 }
5153         }
5154         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5155         {
5156             \int_set:Nn \l_@@_initial_j_int { #2 }
5157             \bool_set_true:N \l_@@_initial_open_bool
5158         }
5159     }
5160     \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5161     {
5162         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5163         \bool_set_true:N \l_@@_final_open_bool
5164     }
5165     {
5166         \cs_if_exist:cTF
5167         {
5168             pgf @ sh @ ns @ \@@_env:
5169             - \int_use:N \l_@@_final_i_int
5170             - \int_eval:n { #2 + #3 }
5171         }
5172         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5173         {
5174             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5175             \bool_set_true:N \l_@@_final_open_bool
5176         }
5177     }
5178     \group_begin:
5179     \@@_open_shorten:
5180     \int_if_zero:nTF { #1 }
5181     { \color { nicematrix-first-row } }
5182     {
5183         \int_compare:nNnT { #1 } = \g_@@_row_total_int
5184         { \color { nicematrix-last-row } }
5185     }
5186
5187     \keys_set:nn { nicematrix / xdots } { #4 }
5188     \@@_color:o \l_@@_xdots_color_tl
5189     \@@_actually_draw_Ldots:
5190     \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).

```

5191 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5192   { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5193 }

5194 \hook_gput_code:nnn { begindocument } { . }
5195 {
5196   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5197   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5198   \cs_new_protected:Npn \@@_Vdotsfor:
5199     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5200   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5201     {
5202       \bool_gset_true:N \g_@@_empty_cell_bool
5203       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5204         {
5205           \@@_Vdotsfor:nnnn
5206             { \int_use:N \c@iRow }
5207             { \int_use:N \c@jCol }
5208             { #2 }
5209             {
5210               #1 , #3 ,
5211               down = \exp_not:n { #4 } ,
5212               up = \exp_not:n { #5 } ,
5213               middle = \exp_not:n { #6 }
5214             }
5215           }
5216         }
5217     }

5218 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5219 {
5220   \bool_set_false:N \l_@@_initial_open_bool
5221   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5222 \int_set:Nn \l_@@_initial_j_int { #2 }
5223 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it’s a bit more complicated.

```

5224 \int_compare:nNnTF { #1 } = \c_one_int
5225 {
5226   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5227   \bool_set_true:N \l_@@_initial_open_bool
5228 }
5229 {
5230   \cs_if_exist:cTF
5231     {
5232       pgf @ sh @ ns @ \@@_env:
5233       - \int_eval:n { #1 - 1 }
5234       - \int_use:N \l_@@_initial_j_int
5235     }
5236     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5237     {
5238       \int_set:Nn \l_@@_initial_i_int { #1 }
5239       \bool_set_true:N \l_@@_initial_open_bool
5240     }
5241   }
5242 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5243 {

```

```

5244     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5245     \bool_set_true:N \l_@@_final_open_bool
5246   }
5247   {
5248     \cs_if_exist:cTF
5249     {
5250       pgf @ sh @ ns @ \@@_env:
5251       - \int_eval:n { #1 + #3 }
5252       - \int_use:N \l_@@_final_j_int
5253     }
5254     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5255     {
5256       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5257       \bool_set_true:N \l_@@_final_open_bool
5258     }
5259   }

5260 \group_begin:
5261 \@@_open_shorten:
5262 \int_if_zero:nTF { #2 }
5263   { \color { nicematrix-first-col } }
5264   {
5265     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5266     { \color { nicematrix-last-col } }
5267   }
5268 \keys_set:nn { nicematrix / xdots } { #4 }
5269 \@@_color:o \l_@@_xdots_color_tl
5270 \@@_actually_draw_Vdots:
5271 \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).

```

5272     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5273     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5274   }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5275 \NewDocumentCommand \@@_rotate: { 0 { } }
5276   {
5277     \peek_remove_spaces:n
5278     {
5279       \bool_gset_true:N \g_@@_rotate_bool
5280       \keys_set:nn { nicematrix / rotate } { #1 }
5281     }
5282   }

5283 \keys_define:nn { nicematrix / rotate }
5284   {
5285     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5286     c .value_forbidden:n = true ,
5287     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5288   }

```

19 The command `\line` accessible in `code-after`

In the `\CodeAfter`, the command `\@@_line:n` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format i - j) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format i - j , our command applies the command `\int_eval:n` to i and j ;
- If not (that is to say, when it's a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹³

```

5289 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5290 {
5291   \tl_if_empty:nTF { #2 }
5292     { #1 }
5293     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5294 }
5295 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5296 { \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).

```

5297 \hook_gput_code:nnn { begindocument } { . }
5298 {
5299   \cs_set_nopar:Npn \l_@@_argspec_tl
5300     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5301   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5302   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5303     {
5304       \group_begin:
5305       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5306       \@@_color:o \l_@@_xdots_color_tl
5307       \use:e
5308       {
5309         \@@_line_i:nn
5310         { \@@_double_int_eval:n #2 - \q_stop }
5311         { \@@_double_int_eval:n #3 - \q_stop }
5312       }
5313       \group_end:
5314     }
5315 }
5316 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5317 {
5318   \bool_set_false:N \l_@@_initial_open_bool
5319   \bool_set_false:N \l_@@_final_open_bool
5320   \bool_lazy_or:nnTF
5321     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5322     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5323     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
5324   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5325 }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5326 \hook_gput_code:nnn { begindocument } { . }
5327 {
5328   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5329   {

```

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:`.

```

5330     \c_@@_pgfortikzpicture_tl
5331     \@@_draw_line_iii:nn { #1 } { #2 }
5332     \c_@@_endpgfortikzpicture_tl
5333   }
5334 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5335 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5336 {
5337   \pgfrememberpicturepositiononpagetrue
5338   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5339   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5340   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5341   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5342   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5343   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5344   \@@_draw_line:
5345 }

```

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_than: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_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

```

5346 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5347 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
5348 \cs_new:Npn \@@_if_col_greater_than:nn #1 #2
5349 { \int_compare:nNnF { \c@jCol } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```

5350 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5351 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5352 {
5353   \tl_gput_right:Ne \g_@@_row_style_tl
5354   {

```


Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5355     \exp_not:N
5356     \@@_if_row_less_than:nn
5357     { \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`).

```

5358     {
5359     \exp_not:N
5360     \@@_if_col_greater_than:nn
5361     { \int_eval:n { \c@jCol } }
5362     { \exp_not:n { #1 } \scan_stop: }
5363     }
5364   }
5365 }

```

```

5366 \keys_define:nn { nicematrix / RowStyle }
5367 {
5368   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5369   cell-space-top-limit .value_required:n = true ,
5370   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5371   cell-space-bottom-limit .value_required:n = true ,
5372   cell-space-limits .meta:n =
5373   {
5374     cell-space-top-limit = #1 ,
5375     cell-space-bottom-limit = #1 ,
5376   } ,
5377   color .tl_set:N = \l_@@_color_tl ,
5378   color .value_required:n = true ,
5379   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5380   bold .default:n = true ,
5381   nb-rows .code:n =
5382     \str_if_eq:eeTF { #1 } { * }
5383     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5384     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5385   nb-rows .value_required:n = true ,
5386   fill .tl_set:N = \l_@@_fill_tl ,
5387   fill .value_required:n = true ,
5388   opacity .tl_set:N = \l_@@_opacity_tl ,
5389   opacity .value_required:n = true ,
5390   rowcolor .tl_set:N = \l_@@_fill_tl ,
5391   rowcolor .value_required:n = true ,
5392   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5393   rounded-corners .default:n = 4 pt ,
5394   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5395 }

```

```

5396 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5397 {
5398   \group_begin:
5399   \tl_clear:N \l_@@_fill_tl
5400   \tl_clear:N \l_@@_opacity_tl
5401   \tl_clear:N \l_@@_color_tl
5402   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5403   \dim_zero:N \l_@@_rounded_corners_dim
5404   \dim_zero:N \l_tmpa_dim
5405   \dim_zero:N \l_tmpb_dim
5406   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` (of its alias `fill`) has been used.

```

5407   \tl_if_empty:NF \l_@@_fill_tl
5408   {

```

```

5409     \@@_add_opacity_to_fill:
5410     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5411     {

```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```

5412         \int_compare:nNnTF \c@jCol > \c_one_int
5413         {

```

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*.

```

5414             \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5415             { \int_use:N \c@iRow - \int_use:N \c@jCol }
5416             { \int_use:N \c@iRow - * }
5417             { \dim_use:N \l_@@_rounded_corners_dim }

```

Then, the other rows (if there are several rows).

```

5418                 \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5419                 { \@@_rounded_from_row:n { \c@iRow + 1 } }
5420             }

```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```

5421         { \@@_rounded_from_row:n { \c@iRow } }
5422     }
5423 }
5424 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5425     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5426     {
5427         \@@_put_in_row_style:e
5428         {
5429             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5430             {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5431                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5432                 { \dim_use:N \l_tmpa_dim }
5433             }
5434         }
5435     }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5436     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5437     {
5438         \@@_put_in_row_style:e
5439         {
5440             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5441             {
5442                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5443                 { \dim_use:N \l_tmpb_dim }
5444             }
5445         }
5446     }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5447     \tl_if_empty:NF \l_@@_color_tl
5448     {
5449         \@@_put_in_row_style:e
5450         {
5451             \mode_leave_vertical:
5452             \@@_color:n { \l_@@_color_tl }
5453         }
5454     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5455   \bool_if:NT \l_@@_bold_row_style_bool
5456   {
5457     \@@_put_in_row_style:n
5458     {
5459       \exp_not:n
5460       {
5461         \if_mode_math:
5462         \c_math_toggle_token
5463         \bfseries \boldmath
5464         \c_math_toggle_token
5465         \else:
5466         \bfseries \boldmath
5467         \fi:
5468       }
5469     }
5470   }
5471   \group_end:
5472   \g_@@_row_style_tl
5473   \ignorespaces
5474 }

```

The following commande must *not* be protected.

```

5475 \cs_new:Npn \@@_rounded_from_row:n #1
5476 {
5477   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “- 1” is *not* a subtraction.

```

5478   { \int_eval:n { #1 } - 1 }
5479   {
5480     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5481     - \exp_not:n { \int_use:N \c@jCol }
5482   }
5483   { \dim_use:N \l_@@_rounded_corners_dim }
5484 }

```

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).

```

5485 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5486 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5487 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5488 {

```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```

5489 \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`.

```

5490 \str_if_in:nnF { #1 } { !! }
5491 {
5492 \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

5493 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5494 }
5495 \int_if_zero:nTF \l_tmpa_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5496 {
5497 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5498 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5499 }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5500 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5501 }

```

The following command must be used within a `\pgfpicture`.

```

5502 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5503 {
5504 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5505 {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5506 \group_begin:
5507 \pgfsetcornersarced
5508 {
5509 \pgfpoint
5510 { \l_@@_tab_rounded_corners_dim }
5511 { \l_@@_tab_rounded_corners_dim }
5512 }

```

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.

```

5513 \bool_if:NTF \l_@@_hvlines_bool
5514 {
5515 \pgfpathrectanglecorners
5516 {
5517 \pgfpointadd
5518 { \@@_qpoint:n { row-1 } }
5519 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5520 }
5521 {
5522 \pgfpointadd
5523 {
5524 \@@_qpoint:n
5525 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5526 }

```

```

5527         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5528     }
5529 }
5530 {
5531     \pgfpathrectanglecorners
5532     { \@@_qpoint:n { row-1 } }
5533     {
5534         \pgfpointadd
5535         {
5536             \@@_qpoint:n
5537             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5538         }
5539         { \pgfpoint \c_zero_dim \arrayrulewidth }
5540     }
5541 }
5542 \pgfusepath { clip }
5543 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5544     }
5545 }

```

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`).

```

5546 \cs_new_protected:Npn \@@_actually_color:
5547 {
5548     \pgfpicture
5549     \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.

```

5550     \@@_clip_with_rounded_corners:
5551     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5552     {
5553         \int_compare:nNnTF { ##1 } = \c_one_int
5554         {
5555             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5556             \use:c { g_@@_color _ 1 _tl }
5557             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5558         }
5559         {
5560             \begin { pgfscope }
5561                 \@@_color_opacity ##2
5562                 \use:c { g_@@_color _ ##1 _tl }
5563                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5564                 \pgfusepath { fill }
5565             \end { pgfscope }
5566         }
5567     }
5568 \endpgfpicture
5569 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5570 \cs_new_protected:Npn \@@_color_opacity
5571 {
5572     \peek_meaning:NTF [
5573         { \@@_color_opacity:w }
5574         { \@@_color_opacity:w [ ] }
5575     }

```

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.

```

5576 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5577 {
5578   \tl_clear:N \l_tmpa_tl
5579   \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.
5580   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5581   \tl_if_empty:NTF \l_tmpb_tl
5582     { \@declaredcolor }
5583     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5584 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5585 \keys_define:nn { nicematrix / color-opacity }
5586 {
5587   opacity .tl_set:N          = \l_tmpa_tl ,
5588   opacity .value_required:n = true
5589 }

5590 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5591 {
5592   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5593   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5594   \@@_cartesian_path:
5595 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5596 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5597 {
5598   \tl_if_blank:nF { #2 }
5599   {
5600     \@@_add_to_colors_seq:en
5601     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5602     { \@@_cartesian_color:nn { #3 } { - } }
5603   }
5604 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5605 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5606 {
5607   \tl_if_blank:nF { #2 }
5608   {
5609     \@@_add_to_colors_seq:en
5610     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5611     { \@@_cartesian_color:nn { - } { #3 } }
5612   }
5613 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5614 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5615 {
5616   \tl_if_blank:nF { #2 }
5617   {
5618     \@@_add_to_colors_seq:en
5619     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5620     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5621   }
5622 }

```

The last argument is the radius of the corners of the rectangle.

```

5623 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5624 {
5625   \tl_if_blank:nF { #2 }
5626   {
5627     \@@_add_to_colors_seq:en
5628     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5629     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5630   }
5631 }

```

The last argument is the radius of the corners of the rectangle.

```

5632 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5633 {
5634   \@@_cut_on_hyphen:w #1 \q_stop
5635   \tl_clear_new:N \l_@@_tmpc_tl
5636   \tl_clear_new:N \l_@@_tmpd_tl
5637   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5638   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5639   \@@_cut_on_hyphen:w #2 \q_stop
5640   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5641   \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`.

```

5642   \@@_cartesian_path:n { #3 }
5643 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5644 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5645 {
5646   \clist_map_inline:nn { #3 }
5647   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5648 }

```

```

5649 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5650 {
5651   \int_step_inline:nn \c@iRow
5652   {
5653     \int_step_inline:nn \c@jCol
5654     {
5655       \int_if_even:nTF { #####1 + ##1 }
5656       { \@@_cellcolor [ #1 ] { #2 } }
5657       { \@@_cellcolor [ #1 ] { #3 } }
5658     }
5659   }
5660 }
5661 }

```

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”.

```

5662 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5663 {
5664   \@@_rectanglecolor [ #1 ] { #2 }
5665   { 1 - 1 }
5666   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5667 }

```

```

5668 \keys_define:nn { nicematrix / rowcolors }
5669 {
5670   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5671   respect-blocks .default:n = true ,
5672   cols .tl_set:N = \l_@@_cols_tl ,
5673   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5674   restart .default:n = true ,
5675   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5676 }

```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package `xcolor` (with the option `table`). However, the command `\rowcolors` of `nicematrix` has *not* the optional argument of the command `\rowcolors` of `xcolor`.

Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.

In `nicematrix`, the command `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

#1 (optional) is the color space; **#2** is a list of intervals of rows; **#3** is the list of colors; **#4** is for the optional list of pairs *key=value*.

```

5677 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5678 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5679   \group_begin:
5680   \seq_clear_new:N \l_@@_colors_seq
5681   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5682   \tl_clear_new:N \l_@@_cols_tl
5683   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5684   \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).

```

5685   \int_zero_new:N \l_@@_color_int
5686   \int_set_eq:NN \l_@@_color_int \c_one_int
5687   \bool_if:NT \l_@@_respect_blocks_bool
5688   {

```

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`).

```

5689     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5690     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5691     { \@@_not_in_exterior_p:nnnnn ##1 }
5692   }
5693   \pgfpicture
5694   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5695   \clist_map_inline:nn { #2 }
5696   {
5697     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5698     \tl_if_in:NnTF \l_tmpa_tl { - }
5699     { \@@_cut_on_hyphen:w ##1 \q_stop }
5700     { \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.

```

5701     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5702     \int_set:Nn \l_@@_color_int
5703     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5704     \int_zero_new:N \l_@@_tmpc_int
5705     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5706     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5707     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5708     \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).

```

5709     \bool_if:NT \l_@@_respect_blocks_bool
5710     {
5711         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5712         { \@@_intersect_our_row_p:nnnnn #####1 }
5713         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5714     }
5715     \tl_set:No \l_@@_rows_tl
5716     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5717     \tl_clear_new:N \l_@@_color_tl
5718     \tl_set:Ne \l_@@_color_tl
5719     {
5720         \@@_color_index:n
5721         {
5722             \int_mod:nn
5723             { \l_@@_color_int - 1 }
5724             { \seq_count:N \l_@@_colors_seq }
5725             + 1
5726         }
5727     }
5728     \tl_if_empty:NF \l_@@_color_tl
5729     {
5730         \@@_add_to_colors_seq:ee
5731         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5732         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5733     }
5734     \int_incr:N \l_@@_color_int
5735     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5736 }
5737 }
5738 \endpgfpicture
5739 \group_end:
5740 }

```

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.

```

5741 \cs_new:Npn \@@_color_index:n #1
5742 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5743     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5744     { \@@_color_index:n { #1 - 1 } }
5745     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5746 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5747 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5748 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5749 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5750 {
5751     \int_compare:nNnT { #3 } > \l_tmpb_int
5752     { \int_set:Nn \l_tmpb_int { #3 } }
5753 }

```

```

5754 \prg_new_conditional:Nnn \@@_not_in_exterior:nmnnn p
5755 {
5756   \int_if_zero:nTF { #4 }
5757   \prg_return_false:
5758   {
5759     \int_compare:nNnTF { #2 } > \c@jCol
5760     \prg_return_false:
5761     \prg_return_true:
5762   }
5763 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5764 \prg_new_conditional:Nnn \@@_intersect_our_row:nmnnn p
5765 {
5766   \int_compare:nNnTF { #1 } > \l_tmpa_int
5767   \prg_return_false:
5768   {
5769     \int_compare:nNnTF \l_tmpa_int > { #3 }
5770     \prg_return_false:
5771     \prg_return_true:
5772   }
5773 }

```

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:nmn (used in \@@_rectanglecolor, itself used in \@@_cellcolor).

```

5774 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5775 {
5776   \dim_compare:nNnTF { #1 } = \c_zero_dim
5777   {
5778     \bool_if:NTF
5779     \l_@@_nocolor_used_bool
5780     \@@_cartesian_path_normal_ii:
5781     {
5782       \clist_if_empty:NTF \l_@@_corners_cells_clist
5783       { \@@_cartesian_path_normal_i:n { #1 } }
5784       \@@_cartesian_path_normal_ii:
5785     }
5786   }
5787   { \@@_cartesian_path_normal_i:n { #1 } }
5788 }

```

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.

```

5789 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5790 {
5791   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5792   \clist_map_inline:Nn \l_@@_cols_tl
5793   {
5794     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5795     \tl_if_in:NnTF \l_tmpa_tl { - }
5796     { \@@_cut_on_hyphen:w ##1 \q_stop }
5797     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5798     \tl_if_empty:NTF \l_tmpa_tl
5799     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5800     {

```

```

5801     \str_if_eq:eeT \l_tmpa_tl { * }
5802     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5803   }
5804   \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5805   { \@@_error:n { Invalid~col~number } }
5806   \tl_if_empty:NTF \l_tmpb_tl
5807   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5808   {
5809     \str_if_eq:eeT \l_tmpb_tl { * }
5810     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5811   }
5812   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5813   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5814   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5815   \@@_qpoint:n { col - \l_tmpa_tl }
5816   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5817   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5818   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5819   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5820   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5821   \clist_map_inline:Nn \l_@@_rows_tl
5822   {
5823     \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5824     \tl_if_in:NnTF \l_tmpa_tl { - }
5825     { \@@_cut_on_hyphen:w ####1 \q_stop }
5826     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5827     \tl_if_empty:NTF \l_tmpa_tl
5828     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5829     {
5830       \str_if_eq:eeT \l_tmpa_tl { * }
5831       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5832     }
5833     \tl_if_empty:NTF \l_tmpb_tl
5834     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5835     {
5836       \str_if_eq:eeT \l_tmpb_tl { * }
5837       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5838     }
5839     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5840     { \@@_error:n { Invalid~row~number } }
5841     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5842     { \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`.

```

5843     \cs_if_exist:cF
5844     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5845     {
5846       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5847       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5848       \@@_qpoint:n { row - \l_tmpa_tl }
5849       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5850       \pgfpathrectanglecorners
5851       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5852       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5853     }
5854   }
5855 }
5856 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5857 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5858 {
5859   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5860   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5861   \clist_map_inline:Nn \l_@@_cols_tl
5862   {
5863     \@@_qpoint:n { col - ##1 }
5864     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5865       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5866       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5867     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5868     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5869     \clist_map_inline:Nn \l_@@_rows_tl
5870     {
5871       \@@_if_in_corner:nF { #####1 - ##1 }
5872       {
5873         \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5874         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5875         \@@_qpoint:n { row - #####1 }
5876         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5877         \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5878         {
5879           \pgfpathrectanglecorners
5880             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5881             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5882         }
5883       }
5884     }
5885   }
5886 }

```

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`).

```

5887 \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

```

5888 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5889 {
5890   \bool_set_true:N \l_@@_nocolor_used_bool
5891   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5892   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5893   \clist_map_inline:Nn \l_@@_rows_tl
5894   {
5895     \clist_map_inline:Nn \l_@@_cols_tl
5896     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5897   }
5898 }

```

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.

```

5899 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5900 {
5901   \clist_set_eq:NN \l_tmpa_clist #1

```

```

5902 \clist_clear:N #1
5903 \clist_map_inline:Nn \l_tmpa_clist
5904 {
5905   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5906   \tl_if_in:NnTF \l_tmpa_tl { - }
5907     { \@@_cut_on_hyphen:w ##1 \q_stop }
5908     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5909   \bool_lazy_or:nnT
5910     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5911     { \tl_if_blank_p:o \l_tmpa_tl }
5912     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5913   \bool_lazy_or:nnT
5914     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5915     { \tl_if_blank_p:o \l_tmpb_tl }
5916     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5917   \int_compare:nNnT \l_tmpb_tl > #2
5918     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5919   \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5920     { \clist_put_right:Nn #1 { ####1 } }
5921 }
5922 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5923 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5924 {
5925   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5926   {

```

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`).

```

5927     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5928     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5929   }
5930   \ignorespaces
5931 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5932 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5933 {
5934   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5935   {
5936     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5937     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5938     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5939   }
5940   \ignorespaces
5941 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5942 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5943 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5944 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5945 {
5946   \peek_remove_spaces:n
5947   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5948 }

```

```

5949 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5950 {

```

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`.

```

5951 \seq_gclear:N \g_tmpa_seq
5952 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5953 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5954 \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`.

```

5955 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5956 {
5957   { \int_use:N \c@iRow }
5958   { \exp_not:n { #1 } }
5959   { \exp_not:n { #2 } }
5960   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5961 }
5962 }

```

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`).

```

5963 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5964 {
5965   \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).

```

5966   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5967   {
5968     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5969     {
5970       \@@_rowlistcolors
5971       [ \exp_not:n { #2 } ]
5972       { #1 - \int_eval:n { \c@iRow - 1 } }
5973       { \exp_not:n { #3 } }
5974       [ \exp_not:n { #4 } ]
5975     }
5976   }
5977 }

```

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.

```

5978 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5979 {
5980   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5981   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5982   \seq_gclear:N \g_@@_rowlistcolors_seq
5983 }

```

```

5984 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nmnn #1 #2 #3 #4
5985 {
5986   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5987   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5988 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is written in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5989 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5990 {

```

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).

```

5991   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5992   {

```

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).

```

5993     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5994     {
5995       \exp_not:N \columncolor [ #1 ]
5996       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5997     }
5998   }
5999 }

```

```

6000 \hook_gput_code:nnn { begindocument } { . }
6001 {
6002   \IfPackageLoadedTF { colortbl }
6003   {
6004     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6005     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6006     \cs_new_protected:Npn \@@_revert_colortbl:
6007     {
6008       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6009       {
6010         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6011         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6012       }
6013     }
6014   }
6015   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6016 }

```

```

6017 \cs_new_protected:Npn \@@_EmptyColumn:n #1
6018 {
6019   \clist_map_inline:nn { #1 }
6020   {
6021     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6022     { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6023     \columncolor { nocolor } { ##1 }
6024   }
6025 }

```

```

6026 \cs_new_protected:Npn \@@_EmptyRow:n #1
6027 {
6028   \clist_map_inline:nn { #1 }
6029   {
6030     \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq

```

```

6031         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6032     \rowcolor { nocolor } { ##1 }
6033 }
6034 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

6035 \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`.

```

6036 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6037 {
6038     \int_if_zero:nTF \l_@@_first_col_int
6039     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6040     {
6041         \int_if_zero:nTF \c@jCol
6042         {
6043             \int_compare:nNnF \c@iRow = { -1 }
6044             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6045         }
6046         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6047     }
6048 }

```

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.

```

6049 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6050 {
6051     \int_if_zero:nF \c@iRow
6052     {
6053         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6054         {
6055             \int_compare:nNnT \c@jCol > \c_zero_int
6056             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6057         }
6058     }
6059 }

```

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`.

```

6060 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6061 {
6062     \IfPackageLoadedTF { tikz }

```



```

6063     {
6064         \IfPackageLoadedTF { booktabs }
6065         { #2 }
6066         { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6067     }
6068     { \@@_error:nn { TopRule~without~tikz } { #1 } }
6069 }

6070 \NewExpandableDocumentCommand { \@@_TopRule } { }
6071 { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }

6072 \cs_new:Npn \@@_TopRule_i:
6073 {
6074     \noalign \bgroup
6075     \peek_meaning:NTF [
6076     { \@@_TopRule_ii: }
6077     { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6078 }

6079 \NewDocumentCommand \@@_TopRule_ii: { o }
6080 {
6081     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6082     {
6083         \@@_hline:n
6084         {
6085             position = \int_eval:n { \c@iRow + 1 } ,
6086             tikz =
6087             {
6088                 line-width = #1 ,
6089                 yshift = 0.25 \arrayrulewidth ,
6090                 shorten~< = - 0.5 \arrayrulewidth
6091             } ,
6092             total-width = #1
6093         }
6094     }
6095     \skip_vertical:n { \belowrulesep + #1 }
6096     \egroup
6097 }

6098 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6099 { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }

6100 \cs_new:Npn \@@_BottomRule_i:
6101 {
6102     \noalign \bgroup
6103     \peek_meaning:NTF [
6104     { \@@_BottomRule_ii: }
6105     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6106 }

6107 \NewDocumentCommand \@@_BottomRule_ii: { o }
6108 {
6109     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6110     {
6111         \@@_hline:n
6112         {
6113             position = \int_eval:n { \c@iRow + 1 } ,
6114             tikz =
6115             {
6116                 line-width = #1 ,
6117                 yshift = 0.25 \arrayrulewidth ,
6118                 shorten~< = - 0.5 \arrayrulewidth
6119             } ,
6120             total-width = #1 ,
6121         }
6122     }
6123     \skip_vertical:N \aboverulesep

```

```

6124     \@@_create_row_node_i:
6125     \skip_vertical:n { #1 }
6126     \egroup
6127 }

6128 \NewExpandableDocumentCommand { \@@_MidRule } { }
6129 { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }

6130 \cs_new:Npn \@@_MidRule_i:
6131 {
6132     \noalign \bgroup
6133     \peek_meaning:NTF [
6134     { \@@_MidRule_ii: }
6135     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6136 }

6137 \NewDocumentCommand \@@_MidRule_ii: { o }
6138 {
6139     \skip_vertical:N \aboverulesep
6140     \@@_create_row_node_i:
6141     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6142     {
6143         \@@_hline:n
6144         {
6145             position = \int_eval:n { \c@iRow + 1 } ,
6146             tikz =
6147             {
6148                 line~width = #1 ,
6149                 yshift = 0.25 \arrayrulewidth ,
6150                 shorten~< = - 0.5 \arrayrulewidth
6151             } ,
6152             total~width = #1 ,
6153         }
6154     }
6155     \skip_vertical:n { \belowrulesep + #1 }
6156     \egroup
6157 }

```

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.

```

6158 \keys_define:nn { nicematrix / Rules }
6159 {
6160     position .int_set:N = \l_@@_position_int ,
6161     position .value_required:n = true ,
6162     start .int_set:N = \l_@@_start_int ,
6163     end .code:n =
6164     \bool_lazy_or:nnTF
6165     { \tl_if_empty_p:n { #1 } }
6166     { \str_if_eq_p:ee { #1 } { last } }
6167     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6168     { \int_set:Nn \l_@@_end_int { #1 } }
6169 }

```

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.

```

6170 \keys_define:nm { nicematrix / RulesBis }
6171 {
6172   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6173   multiplicity .initial:n = 1 ,
6174   dotted .bool_set:N = \l_@@_dotted_bool ,
6175   dotted .initial:n = false ,
6176   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]`.

```

6177   color .code:n =
6178     \@@_set_CT@arc@:n { #1 }
6179     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6180   color .value_required:n = true ,
6181   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6182   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.

```

6183   tikz .code:n =
6184     \IfPackageLoadedTF { tikz }
6185       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6186       { \@@_error:n { tikz-without-tikz } } ,
6187   tikz .value_required:n = true ,
6188   total-width .dim_set:N = \l_@@_rule_width_dim ,
6189   total-width .value_required:n = true ,
6190   width .meta:n = { total-width = #1 } ,
6191   unknown .code:n = \@@_error:n { Unknown-key-for-RulesBis }
6192 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6193 \cs_new_protected:Npn \@@_vline:n #1
6194 {

```

The group is for the options.

```

6195   \group_begin:
6196   \int_set_eq:NN \l_@@_end_int \c@iRow
6197   \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).

```

6198   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6199     \@@_vline_i:
6200   \group_end:
6201 }
6202 \cs_new_protected:Npn \@@_vline_i:
6203 {

```

`\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`.

```

6204   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6205   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6206     \l_tmpa_tl
6207   {

```

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.

```

6208     \bool_gset_true:N \g_tmpa_bool
6209     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6210       { \@@_test_vline_in_block:nnnnn ##1 }
6211     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6212       { \@@_test_vline_in_block:nnnnn ##1 }
6213     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6214       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6215     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6216     \bool_if:NTF \g_tmpa_bool
6217       {
6218         \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.

```

6219         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6220       }
6221     {
6222       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6223       {
6224         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6225         \@@_vline_ii:
6226         \int_zero:N \l_@@_local_start_int
6227       }
6228     }
6229   }
6230 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6231 {
6232   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6233   \@@_vline_ii:
6234 }
6235 }

6236 \cs_new_protected:Npn \@@_test_in_corner_v:
6237 {
6238   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6239   {
6240     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6241     { \bool_set_false:N \g_tmpa_bool }
6242   }
6243   {
6244     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6245     {
6246       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6247       { \bool_set_false:N \g_tmpa_bool }
6248       {
6249         \@@_if_in_corner:nT
6250           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6251           { \bool_set_false:N \g_tmpa_bool }
6252       }
6253     }
6254   }
6255 }

6256 \cs_new_protected:Npn \@@_vline_ii:
6257 {
6258   \tl_clear:N \l_@@_tikz_rule_tl
6259   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```

```

6260 \bool_if:NTF \l_@@_dotted_bool
6261   \@@_vline_iv:
6262   {
6263     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6264     \@@_vline_iii:
6265     \@@_vline_v:
6266   }
6267 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6268 \cs_new_protected:Npn \@@_vline_iii:
6269 {
6270   \pgfpicture
6271   \pgfrememberpicturepositiononpagetrue
6272   \pgf@relevantforpicturesizefalse
6273   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6274   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6275   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6276   \dim_set:Nn \l_tmpb_dim
6277   {
6278     \pgf@x
6279     - 0.5 \l_@@_rule_width_dim
6280     +
6281     ( \arrayrulewidth * \l_@@_multiplicity_int
6282       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6283   }
6284   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6285   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6286   \bool_lazy_all:nT
6287   {
6288     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6289     { \cs_if_exist_p:N \CT@drsc@ }
6290     { ! \tl_if_blank_p:o \CT@drsc@ }
6291   }
6292   {
6293     \group_begin:
6294     \CT@drsc@
6295     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6296     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6297     \dim_set:Nn \l_@@_tmpd_dim
6298     {
6299       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6300       * ( \l_@@_multiplicity_int - 1 )
6301     }
6302     \pgfpathrectanglecorners
6303     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6304     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6305     \pgfusepath { fill }
6306     \group_end:
6307   }
6308   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6309   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6310   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6311   {
6312     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6313     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6314     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6315     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6316   }
6317   \CT@arc@
6318   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6319   \pgfsetrectcap
6320   \pgfusepathqstroke

```

```

6321 \endpgfpicture
6322 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6323 \cs_new_protected:Npn \@@_vline_iv:
6324 {
6325   \pgfpicture
6326   \pgfrememberpicturepositiononpagetrue
6327   \pgf@relevantforpicturesizefalse
6328   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6329   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6330   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6331   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6332   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6333   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6334   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6335   \CT@arc@
6336   \@@_draw_line:
6337   \endpgfpicture
6338 }

```

The following code is for the case when the user uses the key `tikz`.

```

6339 \cs_new_protected:Npn \@@_vline_v:
6340 {
6341   \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).

```

6342   \CT@arc@
6343   \tl_if_empty:NF \l_@@_rule_color_tl
6344   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6345   \pgfrememberpicturepositiononpagetrue
6346   \pgf@relevantforpicturesizefalse
6347   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6348   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6349   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6350   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6351   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6352   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6353   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6354   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6355   ( \l_tmpb_dim , \l_tmpa_dim ) --
6356   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6357   \end {tikzpicture }
6358 }

```

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).

```

6359 \cs_new_protected:Npn \@@_draw_vlines:
6360 {
6361   \int_step_inline:nnn
6362   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6363   {
6364     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6365     \c@jCol
6366     { \int_eval:n { \c@jCol + 1 } }
6367   }
6368   {
6369     \str_if_eq:eeF \l_@@_vlines_clist { all }
6370     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6371     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6372     }
6373 }

```

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}`.

```

6374 \cs_new_protected:Npn \@@_hline:n #1
6375 {

```

The group is for the options.

```

6376     \group_begin:
6377     \int_zero_new:N \l_@@_end_int
6378     \int_set_eq:NN \l_@@_end_int \c@jCol
6379     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6380     \@@_hline_i:
6381     \group_end:
6382 }

6383 \cs_new_protected:Npn \@@_hline_i:
6384 {
6385     \int_zero_new:N \l_@@_local_start_int
6386     \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`.

```

6387     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6388     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6389     \l_tmpb_tl
6390     {

```

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.

```

6391         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6392         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6393         { \@@_test_hline_in_block:nnnnn ##1 }

6394         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6395         { \@@_test_hline_in_block:nnnnn ##1 }

6396         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6397         { \@@_test_hline_in_stroken_block:nnnn ##1 }

6398         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6399         \bool_if:NTF \g_tmpa_bool
6400         {
6401             \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.

```

6402             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6403         }
6404     {
6405         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6406         {
6407             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6408             \@@_hline_ii:
6409             \int_zero:N \l_@@_local_start_int
6410         }
6411     }
6412 }
6413 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6414     {
6415         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6416         \@@_hline_ii:
6417     }
6418 }

6419 \cs_new_protected:Npn \@@_test_in_corner_h:
6420 {
6421     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6422     {
6423         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6424         { \bool_set_false:N \g_tmpa_bool }
6425     }
6426     {
6427         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6428         {
6429             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6430             { \bool_set_false:N \g_tmpa_bool }
6431             {
6432                 \@@_if_in_corner:nT
6433                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6434                 { \bool_set_false:N \g_tmpa_bool }
6435             }
6436         }
6437     }
6438 }

6439 \cs_new_protected:Npn \@@_hline_ii:
6440 {
6441     \tl_clear:N \l_@@_tikz_rule_tl
6442     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6443     \bool_if:NTF \l_@@_dotted_bool
6444     \@@_hline_iv:
6445     {
6446         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6447         \@@_hline_iii:
6448         \@@_hline_v:
6449     }
6450 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6451 \cs_new_protected:Npn \@@_hline_iii:
6452 {
6453     \pgfpicture
6454     \pgfrememberpicturepositiononpagetrue
6455     \pgf@relevantforpicturesizefalse
6456     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6457     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6458     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6459     \dim_set:Nn \l_tmpb_dim
6460     {
6461         \pgf@y
6462         - 0.5 \l_@@_rule_width_dim
6463         +
6464         ( \arrayrulewidth * \l_@@_multiplicity_int
6465           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6466     }
6467     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6468     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6469     \bool_lazy_all:nT
6470     {

```



```

6471     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6472     { \cs_if_exist_p:N \CT@drsc@ }
6473     { ! \tl_if_blank_p:o \CT@drsc@ }
6474   }
6475   {
6476     \group_begin:
6477     \CT@drsc@
6478     \dim_set:Nn \l_@@_tmpd_dim
6479     {
6480       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6481       * ( \l_@@_multiplicity_int - 1 )
6482     }
6483     \pgfpathrectanglecorners
6484     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6485     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6486     \pgfusepathqfill
6487     \group_end:
6488   }
6489   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6490   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6491   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6492   {
6493     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6494     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6495     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6496     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6497   }
6498   \CT@arc@
6499   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6500   \pgfsetrectcap
6501   \pgfusepathqstroke
6502   \endpgfpicture
6503 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6504 \cs_new_protected:Npn \@@_hline_iv:
6505   {
6506     \pgfpicture
6507     \pgfrememberpicturepositiononpagetrue
6508     \pgf@relevantforpicturesizefalse
6509     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6510     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6511     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6512     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6513     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

6514 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6515 {
6516   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6517   \bool_if:NF \g_@@_delims_bool
6518   { \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.

```

6519   \tl_if_eq:NnF \g_@@_left_delim_tl (
6520     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6521   )
6522   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6523   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6524   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6525   {
6526     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6527     \bool_if:NF \g_@@_delims_bool
6528     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6529     \tl_if_eq:NnF \g_@@_right_delim_tl )
6530     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6531   }
6532   \CT@arc@
6533   \@@_draw_line:
6534   \endpgfpicture
6535 }

```

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`).

```

6536 \cs_new_protected:Npn \@@_hline_v:
6537 {
6538   \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).

```

6539   \CT@arc@
6540   \tl_if_empty:NF \l_@@_rule_color_tl
6541   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6542   \pgfrememberpicturepositiononpagetrue
6543   \pgf@relevantforpicturesizefalse
6544   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6545   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6546   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6547   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6548   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6549   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6550   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6551   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6552   ( \l_tmpa_dim , \l_tmpb_dim ) --
6553   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6554   \end { tikzpicture }
6555 }

```

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).

```

6556 \cs_new_protected:Npn \@@_draw_hlines:
6557 {
6558   \int_step_inline:nnn
6559   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6560   {
6561     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6562     \c@iRow
6563     { \int_eval:n { \c@iRow + 1 } }
6564   }
6565   {
6566     \str_if_eq:eeF \l_@@_hlines_clist { all }
6567     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6568     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6569   }
6570 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6571 \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.

```

6572 \cs_set:Npn \@@_Hline_i:n #1
6573 {
6574   \peek_remove_spaces:n
6575   {
6576     \peek_meaning:NTF \Hline
6577     { \@@_Hline_ii:nn { #1 + 1 } }
6578     { \@@_Hline_iii:n { #1 } }
6579   }
6580 }
6581 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6582 \cs_set:Npn \@@_Hline_iii:n #1
6583 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6584 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6585 {
6586   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6587   \skip_vertical:N \l_@@_rule_width_dim
6588   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6589   {
6590     \@@_hline:n
6591     {
6592       multiplicity = #1 ,
6593       position = \int_eval:n { \c@iRow + 1 } ,
6594       total-width = \dim_use:N \l_@@_rule_width_dim ,
6595       #2
6596     }
6597   }
6598   \egroup
6599 }

```

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`).

```

6600 \cs_new_protected:Npn \@@_custom_line:n #1
6601 {
6602   \str_clear_new:N \l_@@_command_str
6603   \str_clear_new:N \l_@@_ccommand_str
6604   \str_clear_new:N \l_@@_letter_str
6605   \tl_clear_new:N \l_@@_other_keys_tl
6606   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6607 \bool_lazy_all:nTF
6608 {
6609   { \str_if_empty_p:N \l_@@_letter_str }
6610   { \str_if_empty_p:N \l_@@_command_str }
6611   { \str_if_empty_p:N \l_@@_ccommand_str }
6612 }
6613 { \@@_error:n { No~letter~and~no~command } }
6614 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6615 }
6616 \keys_define:nm { nicematrix / custom-line }
6617 {
6618   letter .str_set:N = \l_@@_letter_str ,
6619   letter .value_required:n = true ,
6620   command .str_set:N = \l_@@_command_str ,
6621   command .value_required:n = true ,
6622   ccommand .str_set:N = \l_@@_cccommand_str ,
6623   ccommand .value_required:n = true ,
6624 }
6625 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6626 \cs_new_protected:Npn \@@_custom_line_i:n #1
6627 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6628 \bool_set_false:N \l_@@_tikz_rule_bool
6629 \bool_set_false:N \l_@@_dotted_rule_bool
6630 \bool_set_false:N \l_@@_color_bool
6631 \keys_set:nm { nicematrix / custom-line-bis } { #1 }
6632 \bool_if:NT \l_@@_tikz_rule_bool
6633 {
6634   \IfPackageLoadedF { tikz }
6635   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6636   \bool_if:NT \l_@@_color_bool
6637   { \@@_error:n { color-in-custom-line-with-tikz } }
6638 }
6639 \bool_if:NT \l_@@_dotted_rule_bool
6640 {
6641   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6642   { \@@_error:n { key~multiplicity~with~dotted } }
6643 }
6644 \str_if_empty:NF \l_@@_letter_str
6645 {
6646   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6647   { \@@_error:n { Several~letters } }
6648   {
6649     \tl_if_in:NoTF
6650     \c_@@_forbidden_letters_str
6651     \l_@@_letter_str
6652     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6653     {

```

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.

```

6654 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6655 { \@@_v_custom_line:n { #1 } }
6656 }
6657 }
6658 }
6659 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6660 \str_if_empty:NF \l_@@_cccommand_str { \@@_c_custom_line:n { #1 } }
6661 }

```

```

6662 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6663 \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.

```

6664 \keys_define:nn { nicematrix / custom-line-bis }
6665 {
6666   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6667   multiplicity .initial:n = 1 ,
6668   multiplicity .value_required:n = true ,
6669   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6670   color .value_required:n = true ,
6671   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6672   tikz .value_required:n = true ,
6673   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6674   dotted .value_forbidden:n = true ,
6675   total-width .code:n = { } ,
6676   total-width .value_required:n = true ,
6677   width .code:n = { } ,
6678   width .value_required:n = true ,
6679   sep-color .code:n = { } ,
6680   sep-color .value_required:n = true ,
6681   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6682 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6683 \bool_new:N \l_@@_dotted_rule_bool
6684 \bool_new:N \l_@@_tikz_rule_bool
6685 \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`.

```

6686 \keys_define:nn { nicematrix / custom-line-width }
6687 {
6688   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6689   multiplicity .initial:n = 1 ,
6690   multiplicity .value_required:n = true ,
6691   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6692   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6693   \bool_set_true:N \l_@@_total_width_bool ,
6694   total-width .value_required:n = true ,
6695   width .meta:n = { total-width = #1 } ,
6696   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6697 }

```

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`).

```

6698 \cs_new_protected:Npn \@@_h_custom_line:n #1
6699 {

```

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`).

```

6700   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6701   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6702 }

```

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`).

```
6703 \cs_new_protected:Npn \@@_c_custom_line:n #1
6704 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6705   \exp_args:Nc \NewExpandableDocumentCommand
6706     { nicematrix - \l_@@_ccommand_str }
6707     { 0 { } m }
6708     {
6709       \noalign
6710       {
6711         \@@_compute_rule_width:n { #1 , ##1 }
6712         \skip_vertical:n { \l_@@_rule_width_dim }
6713         \clist_map_inline:nn
6714           { ##2 }
6715           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6716       }
6717     }
6718   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6719 }
```

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*.

```
6720 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6721 {
6722   \tl_if_in:nnTF { #2 } { - }
6723     { \@@_cut_on_hyphen:w #2 \q_stop }
6724     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6725   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6726     {
6727       \@@_hline:n
6728       {
6729         #1 ,
6730         start = \l_tmpa_tl ,
6731         end = \l_tmpb_tl ,
6732         position = \int_eval:n { \c@iRow + 1 } ,
6733         total-width = \dim_use:N \l_@@_rule_width_dim
6734       }
6735     }
6736 }

6737 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6738 {
6739   \bool_set_false:N \l_@@_tikz_rule_bool
6740   \bool_set_false:N \l_@@_total_width_bool
6741   \bool_set_false:N \l_@@_dotted_rule_bool
6742   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6743   \bool_if:NF \l_@@_total_width_bool
6744     {
6745       \bool_if:NTF \l_@@_dotted_rule_bool
6746         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6747         {
6748           \bool_if:NF \l_@@_tikz_rule_bool
6749             {
6750               \dim_set:Nn \l_@@_rule_width_dim
6751                 {
6752                   \arrayrulewidth * \l_@@_multiplicity_int
6753                   + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6754                 }
6755             }
6756         }
6757     }
6758 }
```

```

6759 \cs_new_protected:Npn \@@_v_custom_line:n #1
6760 {
6761   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6762   \tl_gput_right:Ne \g_@@_array_preamble_tl
6763     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6764   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6765     {
6766     \@@_vline:n
6767     {
6768       #1 ,
6769       position = \int_eval:n { \c@jCol + 1 } ,
6770       total-width = \dim_use:N \l_@@_rule_width_dim
6771     }
6772   }
6773   \@@_rec_preamble:n
6774 }

6775 \@@_custom_line:n
6776 { 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.

```

6777 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6778 {
6779   \int_compare:nNnT \l_tmpa_tl > { #1 }
6780   {
6781     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6782     {
6783       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6784       {
6785         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6786         { \bool_gset_false:N \g_tmpa_bool }
6787       }
6788     }
6789   }
6790 }

```

The same for vertical rules.

```

6791 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6792 {
6793   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6794   {
6795     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6796     {
6797       \int_compare:nNnT \l_tmpb_tl > { #2 }
6798       {
6799         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6800         { \bool_gset_false:N \g_tmpa_bool }
6801       }
6802     }
6803   }
6804 }

6805 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6806 {
6807   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6808   {
6809     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6810     {

```

```

6811         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6812         { \bool_gset_false:N \g_tmpa_bool }
6813         {
6814             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6815             { \bool_gset_false:N \g_tmpa_bool }
6816         }
6817     }
6818 }
6819 }

6820 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6821 {
6822     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6823     {
6824         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6825         {
6826             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6827             { \bool_gset_false:N \g_tmpa_bool }
6828             {
6829                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6830                 { \bool_gset_false:N \g_tmpa_bool }
6831             }
6832         }
6833     }
6834 }

```

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.

```

6835 \cs_new_protected:Npn \@@_compute_corners:
6836 {
6837     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6838     { \@@_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`).

```

6839     \clist_clear:N \l_@@_corners_cells_clist
6840     \clist_map_inline:Nn \l_@@_corners_clist
6841     {
6842         \str_case:nnF { ##1 }
6843         {
6844             { NW }
6845             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6846             { NE }
6847             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6848             { SW }
6849             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6850             { SE }
6851             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6852         }
6853         { \@@_error:nn { bad-corner } { ##1 } }
6854     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6855     \clist_if_empty:NF \l_@@_corners_cells_clist
6856     {

```


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.

```

6857     \tl_gput_right:Nc \g_@@_aux_tl
6858     {
6859         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6860         { \l_@@_corners_cells_clist }
6861     }
6862 }
6863 }

6864 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6865 {
6866     \int_step_inline:nnn { #1 } { #3 }
6867     {
6868         \int_step_inline:nnn { #2 } { #4 }
6869         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6870     }
6871 }

6872 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6873 {
6874     \cs_if_exist:cTF
6875     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6876     \prg_return_true:
6877     \prg_return_false:
6878 }

```

“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.

```

6879 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6880 {

```

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.

```

6881     \bool_set_false:N \l_tmpa_bool
6882     \int_zero_new:N \l_@@_last_empty_row_int
6883     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6884     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6885     {
6886         \bool_lazy_or:nnTF
6887         {
6888             \cs_if_exist_p:c
6889             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6890         }
6891         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6892         { \bool_set_true:N \l_tmpa_bool }
6893     }

```

```

6894         \bool_if:NF \l_tmpa_bool
6895         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6896     }
6897 }

```

Now, you determine the last empty cell in the row of number 1.

```

6898     \bool_set_false:N \l_tmpa_bool
6899     \int_zero_new:N \l_@@_last_empty_column_int
6900     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6901     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6902     {
6903         \bool_lazy_or:nnTF
6904         {
6905             \cs_if_exist_p:c
6906             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6907         }
6908         { \@@_if_in_block_p:nn { #1 } { ##1 } }
6909         { \bool_set_true:N \l_tmpa_bool }
6910     }
6911     \bool_if:NF \l_tmpa_bool
6912     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6913 }
6914 }

```

Now, we loop over the rows.

```

6915     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6916     {

```

We treat the row number ##1 with another loop.

```

6917         \bool_set_false:N \l_tmpa_bool
6918         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6919         {
6920             \bool_lazy_or:nnTF
6921             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6922             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6923             { \bool_set_true:N \l_tmpa_bool }
6924         }
6925         \bool_if:NF \l_tmpa_bool
6926         {
6927             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6928             \clist_put_right:Nn
6929             \l_@@_corners_cells_clist
6930             { ##1 - #####1 }
6931             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6932         }
6933     }
6934 }
6935 }
6936 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6937 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6938 \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.

```
6939 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6940 \keys_define:nn { nicematrix / NiceMatrixBlock }
6941 {
6942   auto-columns-width .code:n =
6943   {
6944     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6945     \dim_gzero_new:N \g_@@_max_cell_width_dim
6946     \bool_set_true:N \l_@@_auto_columns_width_bool
6947   }
6948 }

6949 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6950 {
6951   \int_gincr:N \g_@@_NiceMatrixBlock_int
6952   \dim_zero:N \l_@@_columns_width_dim
6953   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6954   \bool_if:NT \l_@@_block_auto_columns_width_bool
6955   {
6956     \cs_if_exist:cT
6957     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6958     {
6959       \dim_set:Nn \l_@@_columns_width_dim
6960       {
6961         \use:c
6962         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6963       }
6964     }
6965   }
6966 }
```

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`).

```
6967 {
6968   \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.

```
6969   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6970   {
6971     \bool_if:NT \l_@@_block_auto_columns_width_bool
6972     {
6973       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6974       \iow_shipout:Ne \@mainaux
6975       {
6976         \cs_gset:cpn
6977         { @@ _ 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.

```
6978         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6979       }
6980       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6981     }
6982   }
6983   \ignorespacesafterend
6984 }
```

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).

```

6985 \cs_new_protected:Npn \@@_create_extra_nodes:
6986 {
6987   \bool_if:nTF \l_@@_medium_nodes_bool
6988     {
6989     \bool_if:NTF \l_@@_no_cell_nodes_bool
6990       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
6991       {
6992         \bool_if:NTF \l_@@_large_nodes_bool
6993           \@@_create_medium_and_large_nodes:
6994           \@@_create_medium_nodes:
6995         }
6996       }
6997     {
6998     \bool_if:NT \l_@@_large_nodes_bool
6999     {
7000       \bool_if:NTF \l_@@_no_cell_nodes_bool
7001         { \@@_error:n { extra-nodes~with~no-cell-nodes } }
7002         \@@_create_large_nodes:
7003       }
7004     }
7005   }

```

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`.

```

7006 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
7007 {
7008   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7009     {
7010     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
7011     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
7012     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
7013     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
7014   }
7015   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7016     {
7017     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
7018     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
7019     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
7020     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
7021   }

```

We begin the two nested loops over the rows and the columns of the array.

```

7022   \int_step_variable:nNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7023   {
7024     \int_step_variable:nNn
7025     \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.

```

7026     {
7027       \cs_if_exist:cT
7028       { 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`.

```

7029     {
7030       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7031       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
7032       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7033       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7034       {
7035         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7036         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7037       }

```

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`.

```

7038       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7039       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
7040       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
7041       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7042       {
7043         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
7044         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7045       }
7046     }
7047   }
7048 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7049   \int_step_variable:nNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7050   {
7051     \dim_compare:nNnT
7052     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
7053     {
7054       \@@_qpoint:n { row - \@@_i: - base }
7055       \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
7056       \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
7057     }
7058   }
7059   \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7060   {
7061     \dim_compare:nNnT
7062     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
7063     {
7064       \@@_qpoint:n { col - \@@_j: }
7065       \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
7066       \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
7067     }
7068   }
7069 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7070 \cs_new_protected:Npn \@@_create_medium_nodes:
7071 {
7072   \pgfpicture
7073     \pgfrememberpicturepositiononpagetrue
7074     \pgf@relevantforpicturesizefalse
7075     \@@_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”.

```

7076     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7077     \@@_create_nodes:
7078     \endpgfpicture
7079 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹⁴. However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

```

7080 \cs_new_protected:Npn \@@_create_large_nodes:
7081 {
7082   \pgfpicture
7083     \pgfrememberpicturepositiononpagetrue
7084     \pgf@relevantforpicturesizefalse
7085     \@@_computations_for_medium_nodes:
7086     \@@_computations_for_large_nodes:
7087     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
7088     \@@_create_nodes:
7089   \endpgfpicture
7090 }

7091 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7092 {
7093   \pgfpicture
7094     \pgfrememberpicturepositiononpagetrue
7095     \pgf@relevantforpicturesizefalse
7096     \@@_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”.

```

7097     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7098     \@@_create_nodes:
7099     \@@_computations_for_large_nodes:
7100     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
7101     \@@_create_nodes:
7102   \endpgfpicture
7103 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

7104 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7105 {
7106   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7107   \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`.

```

7108   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7109   {
7110     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7111     {
7112     (
7113         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7114         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7115     )
7116     / 2
7117     }
7118     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7119     { l_@@_row_\@@_i: _ min_dim }
7120 }
7121 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7122 {
7123     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7124     {
7125     (
7126         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7127         \dim_use:c
7128         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7129     )
7130     / 2
7131     }
7132     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7133     { l_@@_column _ \@@_j: _ max _ dim }
7134 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7135     \dim_sub:cn
7136     { l_@@_column _ 1 _ min _ dim }
7137     \l_@@_left_margin_dim
7138     \dim_add:cn
7139     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7140     \l_@@_right_margin_dim
7141 }

```

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).

```

7142 \cs_new_protected:Npn \@@_create_nodes:
7143 {
7144     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7145     {
7146         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7147         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7148         \@@_pgf_rect_node:nnnnn
7149         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7150         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7151         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7152         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7153         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7154         \str_if_empty:NF \l_@@_name_str
7155         {
7156             \pgfnodealias
7157             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7158             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7159         }
7160     }
7161 }

```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{n}{...}{...}` with $n > 1$ was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of n .

```

7162     \seq_map_pairwise_function:NNN
7163     \g_@@_multicolumn_cells_seq
7164     \g_@@_multicolumn_sizes_seq
7165     \@@_node_for_multicolumn:nn
7166   }

7167 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7168 {
7169   \cs_set_nopar:Npn \@@_i: { #1 }
7170   \cs_set_nopar:Npn \@@_j: { #2 }
7171 }

```

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”).

```

7172 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7173 {
7174   \@@_extract_coords_values: #1 \q_stop
7175   \@@_pgf_rect_node:nnnnn
7176   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7177   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7178   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7179   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7180   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7181   \str_if_empty:NF \l_@@_name_str
7182   {
7183     \pgfnodealias
7184     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7185     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7186   }
7187 }

```

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).

```

7188 \keys_define:nn { nicematrix / Block / FirstPass }
7189 {
7190   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7191             \bool_set_true:N \l_@@_p_block_bool ,
7192   j .value_forbidden:n = true ,
7193   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7194   l .value_forbidden:n = true ,
7195   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7196   r .value_forbidden:n = true ,
7197   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7198   c .value_forbidden:n = true ,
7199   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7200   L .value_forbidden:n = true ,
7201   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,

```



```

7202 R .value_forbidden:n = true ,
7203 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7204 C .value_forbidden:n = true ,
7205 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7206 t .value_forbidden:n = true ,
7207 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7208 T .value_forbidden:n = true ,
7209 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7210 b .value_forbidden:n = true ,
7211 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7212 B .value_forbidden:n = true ,
7213 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7214 m .value_forbidden:n = true ,
7215 v-center .meta:n = m ,
7216 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7217 p .value_forbidden:n = true ,
7218 color .code:n =
7219   \@@_color:n { #1 }
7220   \tl_set_rescan:Nnn
7221     \l_@@_draw_tl
7222     { \char_set_catcode_other:N ! }
7223     { #1 } ,
7224 color .value_required:n = true ,
7225 respect-arraystretch .code:n =
7226   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7227 respect-arraystretch .value_forbidden:n = true ,
7228 }

```

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.

```

7229 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7230 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7231 {

```

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).

```

7232   \peek_remove_spaces:n
7233   {
7234     \tl_if_blank:nTF { #2 }
7235     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7236     {
7237       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7238       \@@_Block_i_czech \@@_Block_i
7239       #2 \q_stop
7240     }
7241     { #1 } { #3 } { #4 }
7242   }
7243 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7244 \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`.

```

7245 {
7246   \char_set_catcode_active:N -
7247   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7248 }

```

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.

```
7249 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7250 {
```

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).

```
7251 \bool_lazy_or:nnTF
7252 { \tl_if_blank_p:n { #1 } }
7253 { \str_if_eq_p:ee { * } { #1 } }
7254 { \int_set:Nn \l_tmpa_int { 100 } }
7255 { \int_set:Nn \l_tmpa_int { #1 } }
7256 \bool_lazy_or:nnTF
7257 { \tl_if_blank_p:n { #2 } }
7258 { \str_if_eq_p:ee { * } { #2 } }
7259 { \int_set:Nn \l_tmpb_int { 100 } }
7260 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7261 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7262 {
7263   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7264     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7265     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7266   }
7267   { \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.

```
7268 \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7269 \tl_set:Nc \l_tmpa_tl
7270 {
7271   { \int_use:N \c@iRow }
7272   { \int_use:N \c@jCol }
7273   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7274   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7275 }
```

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).

```
7276 \bool_set_false:N \l_tmpa_bool
7277 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
7278 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7279 \bool_case:nF
7280 {
7281   \l_tmpa_bool { \@@_Block_vii:eennn }
7282   \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.

```

7283     \l_@@_X_bool                               { \@@_Block_v:eennn }
7284     { \tl_if_empty_p:n { #5 } }                 { \@@_Block_v:eennn }
7285     { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7286     { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7287   }
7288   { \@@_Block_v:eennn }
7289   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7290 }

```

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.

```

7291 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7292 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7293 {
7294   \int_gincr:N \g_@@_block_box_int
7295   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7296   {
7297     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7298     {
7299       \@@_actually_diagbox:nnnnnn
7300       { \int_use:N \c@iRow }
7301       { \int_use:N \c@jCol }
7302       { \int_eval:n { \c@iRow + #1 - 1 } }
7303       { \int_eval:n { \c@jCol + #2 - 1 } }
7304       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7305       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7306     }
7307   }
7308   \box_gclear_new:c
7309   { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

7310   \hbox_gset:cn
7311   { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7312   {

```

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).

```

7313     \tl_if_empty:NTF \l_@@_color_tl
7314     { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7315     { \@@_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.

```

7316     \int_compare:nNnT { #1 } = \c_one_int
7317     {
7318         \int_if_zero:nTF \c@iRow
7319         {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\\
-2 & 3 & -4 & 5 & \\\
3 & -4 & 5 & -6 & \\\
-4 & 5 & -6 & 7 & \\\
5 & -6 & 7 & -8 & \\\
\end{bNiceMatrix}$

```

```

7320     \cs_set_eq:NN \Block \@@_NullBlock:
7321     \l_@@_code_for_first_row_tl
7322     }
7323     {
7324     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7325     {
7326     \cs_set_eq:NN \Block \@@_NullBlock:
7327     \l_@@_code_for_last_row_tl
7328     }
7329     }
7330     \g_@@_row_style_tl
7331     }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7332     \@@_reset_arraystretch:
7333     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7334     #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.).

```

7335     \@@_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}`.

```

7336     \bool_if:NTF \l_@@_tabular_bool
7337     {
7338     \bool_lazy_all:nTF
7339     {
7340     { \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.

```
7341         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7342         { ! \g_@@_rotate_bool }
7343     }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7344     {
7345     \use:e
7346     {
```

The `\exp_not:N` is mandatory before `\begin`.

```
7347         \exp_not:N \begin { minipage }%
7348         [ \str_lowercase:o \l_@@_vpos_block_str ]
7349         { \l_@@_col_width_dim }
7350         \str_case:on \l_@@_hpos_block_str
7351         { c \centering r \raggedleft l \raggedright }
7352     }
7353     #5
7354     \end { minipage }
7355 }
```

In the other cases, we use a `{tabular}`.

```
7356     {
7357     \bool_if:NT \c_@@_testphase_table_bool
7358     { \tagpdfsetup { table / tagging = presentation } }
7359     \use:e
7360     {
7361     \exp_not:N \begin { tabular }%
7362     [ \str_lowercase:o \l_@@_vpos_block_str ]
7363     { @ { } \l_@@_hpos_block_str @ { } }
7364     }
7365     #5
7366     \end { tabular }
7367     }
7368 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7369     {
7370     \c_math_toggle_token
7371     \use:e
7372     {
7373     \exp_not:N \begin { array }%
7374     [ \str_lowercase:o \l_@@_vpos_block_str ]
7375     { @ { } \l_@@_hpos_block_str @ { } }
7376     }
7377     #5
7378     \end { array }
7379     \c_math_toggle_token
7380     }
7381 }
```

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).

```
7382     \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.

```
7383     \int_compare:nNnT { #2 } = \c_one_int
7384     {
7385     \dim_gset:Nn \g_@@_blocks_wd_dim
7386     {
```

```

7387         \dim_max:nn
7388         \g_@@_blocks_wd_dim
7389         {
7390         \box_wd:c
7391         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7392         }
7393     }
7394 }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position.

```

7395 \bool_lazy_and:nnT
7396 { \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.

```

7397 { \str_if_empty_p:N \l_@@_vpos_block_str }
7398 {
7399     \dim_gset:Nn \g_@@_blocks_ht_dim
7400     {
7401         \dim_max:nn
7402         \g_@@_blocks_ht_dim
7403         {
7404             \box_ht:c
7405             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7406         }
7407     }
7408     \dim_gset:Nn \g_@@_blocks_dp_dim
7409     {
7410         \dim_max:nn
7411         \g_@@_blocks_dp_dim
7412         {
7413             \box_dp:c
7414             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7415         }
7416     }
7417 }
7418 \seq_gput_right:Ne \g_@@_blocks_seq
7419 {
7420     \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.

```

7421 {
7422     \exp_not:n { #3 } ,
7423     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7424     \bool_if:NT \g_@@_rotate_bool
7425     {
7426         \bool_if:NTF \g_@@_rotate_c_bool
7427         { m }
7428         { \int_compare:nNnT \c_iRow = \l_@@_last_row_int T }
7429     }
7430 }
7431 {
7432     \box_use_drop:c
7433     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7434 }
7435 }
7436 \bool_set_false:N \g_@@_rotate_c_bool
7437 }

```

```

7438 \cs_new:Npn \@_adjust_hpos_rotate:
7439 {
7440   \bool_if:NT \g_@@_rotate_bool
7441   {
7442     \str_set:Ne \l_@@_hpos_block_str
7443     {
7444       \bool_if:NTF \g_@@_rotate_c_bool
7445       { c }
7446       {
7447         \str_case:onF \l_@@_vpos_block_str
7448         { b l B l t r T r }
7449         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7450       }
7451     }
7452   }
7453 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7454 \cs_new_protected:Npn \@_rotate_box_of_block:
7455 {
7456   \box_grotate:cn
7457   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7458   { 90 }
7459   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7460   {
7461     \vbox_gset_top:cn
7462     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7463     {
7464       \skip_vertical:n { 0.8 ex }
7465       \box_use:c
7466       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7467     }
7468   }
7469   \bool_if:NT \g_@@_rotate_c_bool
7470   {
7471     \hbox_gset:cn
7472     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7473     {
7474       \c_math_toggle_token
7475       \vcenter
7476       {
7477         \box_use:c
7478         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7479       }
7480       \c_math_toggle_token
7481     }
7482   }
7483 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. \@_draw_blocks: and above all \@_Block_v:nnnnn).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7484 \cs_generate_variant:Nn \@_Block_v:nnnnn { e e }
7485 \cs_new_protected:Npn \@_Block_v:nnnnn #1 #2 #3 #4 #5
7486 {
7487   \seq_gput_right:Ne \g_@@_blocks_seq
7488   {

```

```

7489     \l_tmpa_tl
7490     { \exp_not:n { #3 } }
7491     {
7492         \bool_if:NTF \l_@@_tabular_bool
7493         {
7494             \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7495         \@@_reset_arraystretch:
7496         \exp_not:n
7497         {
7498             \dim_zero:N \extrarowheight
7499             #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).

```

7500         \bool_if:NT \c_@@_testphase_table_bool
7501         { \tag_stop:n { table } }
7502         \use:e
7503         {
7504             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7505             { @ { } \l_@@_hpos_block_str @ { } }
7506         }
7507         #5
7508         \end { tabular }
7509     }
7510     \group_end:
7511 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7512     {
7513         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7514         \@@_reset_arraystretch:
7515         \exp_not:n
7516         {
7517             \dim_zero:N \extrarowheight
7518             #4
7519             \c_math_toggle_token
7520             \use:e
7521             {
7522                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7523                 { @ { } \l_@@_hpos_block_str @ { } }
7524             }
7525             #5
7526             \end { array }
7527             \c_math_toggle_token
7528         }
7529         \group_end:
7530     }
7531 }
7532 }
7533 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7534 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7535 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7536 {
7537     \seq_gput_right:Ne \g_@@_blocks_seq
7538     {

```



```

7539     \l_tmpa_tl
7540     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7541     { { \exp_not:n { #4 #5 } } }
7542   }
7543 }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7544 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7545 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7546   {
7547     \seq_gput_right:Ne \g_@@_blocks_seq
7548     {
7549       \l_tmpa_tl
7550       { \exp_not:n { #3 } }
7551       { \exp_not:n { #4 #5 } }
7552     }
7553   }

```

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).

```

7554 \keys_define:nn { nicematrix / Block / SecondPass }
7555   {
7556     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7557     ampersand-in-blocks .default:n = true ,
7558     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7559     tikz .code:n =
7560       \IfPackageLoadedTF { tikz }
7561         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7562         { \@@_error:n { tikz~key~without~tikz } } ,
7563     tikz .value_required:n = true ,
7564     fill .code:n =
7565       \tl_set_rescan:Nnn
7566         \l_@@_fill_tl
7567         { \char_set_catcode_other:N ! }
7568         { #1 } ,
7569     fill .value_required:n = true ,
7570     opacity .tl_set:N = \l_@@_opacity_tl ,
7571     opacity .value_required:n = true ,
7572     draw .code:n =
7573       \tl_set_rescan:Nnn
7574         \l_@@_draw_tl
7575         { \char_set_catcode_other:N ! }
7576         { #1 } ,
7577     draw .default:n = default ,
7578     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7579     rounded-corners .default:n = 4 pt ,
7580     color .code:n =
7581       \@@_color:n { #1 }
7582       \tl_set_rescan:Nnn
7583         \l_@@_draw_tl
7584         { \char_set_catcode_other:N ! }
7585         { #1 } ,
7586     borders .clist_set:N = \l_@@_borders_clist ,
7587     borders .value_required:n = true ,
7588     hvlines .meta:n = { vlines , hlines } ,
7589     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7590     vlines .default:n = true ,

```

```

7591 hlines .bool_set:N = \l_@@_hlines_block_bool,
7592 hlines .default:n = true ,
7593 line-width .dim_set:N = \l_@@_line_width_dim ,
7594 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7595 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7596           \bool_set_true:N \l_@@_p_block_bool ,
7597 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7598 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7599 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7600 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7601           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7602 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7603           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7604 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7605           \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7606 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7607 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7608 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7609 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7610 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7611 m .value_forbidden:n = true ,
7612 v-center .meta:n = m ,
7613 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7614 p .value_forbidden:n = true ,
7615 name .tl_set:N = \l_@@_block_name_str ,
7616 name .value_required:n = true ,
7617 name .initial:n = ,
7618 respect-arraystretch .code:n =
7619   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7620 respect-arraystretch .value_forbidden:n = true ,
7621 transparent .bool_set:N = \l_@@_transparent_bool ,
7622 transparent .default:n = true ,
7623 transparent .initial:n = false ,
7624 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7625 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7626 \cs_new_protected:Npn \@@_draw_blocks:
7627 {
7628   \bool_if:NTF \c_@@_recent_array_bool
7629     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7630     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7631   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnn ##1 }
7632 }
7633 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5 #6
7634 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7635   \int_zero_new:N \l_@@_last_row_int
7636   \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”).

```

7637   \int_compare:nNnTF { #3 } > { 98 }

```

```

7638     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7639     { \int_set:Nn \l_@@_last_row_int { #3 } }
7640 \int_compare:nNnTF { #4 } > { 98 }
7641     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7642     { \int_set:Nn \l_@@_last_col_int { #4 } }
7643 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7644 {
7645     \bool_lazy_and:nNTF
7646     \l_@@_preamble_bool
7647     {
7648         \int_compare_p:n
7649         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7650     }
7651     {
7652         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7653         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7654         \@@_msg_redirect_name:nn { columns-not-used } { none }
7655     }
7656     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7657 }
7658 {
7659     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7660     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7661     {
7662         \@@_Block_v:nneenn
7663         { #1 }
7664         { #2 }
7665         { \int_use:N \l_@@_last_row_int }
7666         { \int_use:N \l_@@_last_col_int }
7667         { #5 }
7668         { #6 }
7669     }
7670 }
7671 }

```

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

```

7672 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7673 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7674 {

```

The group is for the keys.

```

7675     \group_begin:
7676     \int_compare:nNnT { #1 } = { #3 }
7677     { \str_set:Nn \l_@@_vpos_block_str { t } }
7678     \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7679     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7680 \bool_lazy_and:nnT
7681     \l_@@_vlines_block_bool
7682     { ! \l_@@_ampersand_bool }
7683     {
7684         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7685         {
7686             \@@_vlines_block:nnn
7687             { \exp_not:n { #5 } }
7688             { #1 - #2 }
7689             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7690         }
7691     }

```

```

7692 \bool_if:NT \l_@@_hlines_block_bool
7693 {
7694   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7695   {
7696     \@@_hlines_block:nnn
7697     { \exp_not:n { #5 } }
7698     { #1 - #2 }
7699     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7700   }
7701 }
7702 \bool_if:NF \l_@@_transparent_bool
7703 {
7704   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7705 }

```

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).

```

7706   \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7707   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7708 }
7709 }

```

```

7710 \tl_if_empty:NF \l_@@_draw_tl
7711 {
7712   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7713   { \@@_error:n { hlines-with-color } }
7714   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7715   {
7716     \@@_stroke_block:nnn

```

#5 are the options

```

7717     { \exp_not:n { #5 } }
7718     { #1 - #2 }
7719     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7720   }
7721   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7722   { { #1 } { #2 } { #3 } { #4 } }
7723 }
7724 \clist_if_empty:NF \l_@@_borders_clist
7725 {
7726   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7727   {
7728     \@@_stroke_borders_block:nnn
7729     { \exp_not:n { #5 } }
7730     { #1 - #2 }
7731     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7732   }
7733 }
7734 \tl_if_empty:NF \l_@@_fill_tl
7735 {
7736   \@@_add_opacity_to_fill:
7737   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7738   {
7739     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7740     { #1 - #2 }
7741     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7742     { \dim_use:N \l_@@_rounded_corners_dim }
7743   }
7744 }
7745 \seq_if_empty:NF \l_@@_tikz_seq
7746 {
7747   \tl_gput_right:Ne \g_nicematrix_code_before_tl

```

```

7748     {
7749         \@_block_tikz:nnnnn
7750         { \seq_use:Nn \l_@@_tikz_seq { , } }
7751         { #1 }
7752         { #2 }
7753         { \int_use:N \l_@@_last_row_int }
7754         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7755     }
7756 }

7757 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7758 {
7759     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7760     {
7761         \@_actually_diagbox:nnnnnn
7762         { #1 }
7763         { #2 }
7764         { \int_use:N \l_@@_last_row_int }
7765         { \int_use:N \l_@@_last_col_int }
7766         { \exp_not:n { ##1 } }
7767         { \exp_not:n { ##2 } }
7768     }
7769 }

```

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.

```

7770 \pgfpicture
7771 \pgfrememberpicturepositiononpagetrue
7772 \pgf@relevantforpicturesizefalse
7773 \@_qpoint:n { row - #1 }
7774 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7775 \@_qpoint:n { col - #2 }
7776 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7777 \@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7778 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7779 \@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7780 \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.

```

7781 \@@_pgf_rect_node:nnnnn
7782 { \@@_env: - #1 - #2 - block }

```

```

7783     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7784 \str_if_empty:NF \l_@@_block_name_str
7785 {
7786     \pgfnodealias
7787     { \@@_env: - \l_@@_block_name_str }
7788     { \@@_env: - #1 - #2 - block }
7789     \str_if_empty:NF \l_@@_name_str
7790     {
7791         \pgfnodealias
7792         { \l_@@_name_str - \l_@@_block_name_str }
7793         { \@@_env: - #1 - #2 - block }
7794     }
7795 }

```

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.

```

7796     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7797     {
7798         \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.

```

7799     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7800     {

```

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.

```

7801         \cs_if_exist:cT
7802         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7803         {
7804             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7805             {
7806                 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7807                 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7808             }
7809         }
7810     }

```

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.

```

7811     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7812     {
7813         \@@_qpoint:n { col - #2 }
7814         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7815     }
7816     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7817     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7818     {
7819         \cs_if_exist:cT
7820         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7821         {
7822             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7823             {
7824                 \pgfpointanchor
7825                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7826                 { east }
7827                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7828             }
7829         }
7830     }
7831     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7832     {

```

```

7833     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7834     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7835   }
7836   \l_@@_pgf_rect_node:nnnnn
7837   { \l_@@_env: - #1 - #2 - block - short }
7838   \l_@@_tmpb_dim \l_@@_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7839 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\l_@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

7840   \bool_if:NT \l_@@_medium_nodes_bool
7841   {
7842     \l_@@_pgf_rect_node:nnn
7843     { \l_@@_env: - #1 - #2 - block - medium }
7844     { \pgfpointanchor { \l_@@_env: - #1 - #2 - medium } { north-west } }
7845     {
7846       \pgfpointanchor
7847       { \l_@@_env:
7848         - \int_use:N \l_@@_last_row_int
7849         - \int_use:N \l_@@_last_col_int - medium
7850       }
7851       { south-east }
7852     }
7853   }
7854   \endpgfpicture

```

```

7855   \bool_if:NTF \l_@@_ampersand_bool
7856   {
7857     \seq_set_split:Nnn \l_@@_tmpa_seq { & } { #6 }
7858     \int_zero_new:N \l_@@_split_int
7859     \int_set:Nn \l_@@_split_int { \seq_count:N \l_@@_tmpa_seq }
7860     \pgfpicture
7861     \pgfrememberpicturepositiononpagetrue
7862     \pgf@relevantforpicturesizefalse
7863
7864     \l_@@_qpoint:n { row - #1 }
7865     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7866     \l_@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7867     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7868     \l_@@_qpoint:n { col - #2 }
7869     \dim_set_eq:NN \l_@@_tmpa_dim \pgf@x
7870     \l_@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7871     \dim_set:Nn \l_@@_tmpb_dim
7872     { ( \pgf@x - \l_@@_tmpa_dim ) / \int_use:N \l_@@_split_int }
7873     \bool_lazy_or:nnT
7874     \l_@@_vlines_block_bool
7875     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7876     {
7877       \int_step_inline:nn { \l_@@_split_int - 1 }
7878       {
7879         \pgfpathmoveto
7880         {
7881           \pgfpoint
7882           { \l_@@_tmpa_dim + ##1 \l_@@_tmpb_dim }
7883           \l_@@_tmpc_dim
7884         }
7885         \pgfpathlineto
7886         {
7887           \pgfpoint
7888           { \l_@@_tmpa_dim + ##1 \l_@@_tmpb_dim }
7889           \l_@@_tmpd_dim
7890         }
7891       }
7892     }
7893     \CT@arc@

```

```

7892         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7893         \pgfsetrectcap
7894         \pgfusepathqstroke
7895     }
7896 }
7897 \@@_qpoint:n { row - #1 - base }
7898 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7899 \int_step_inline:nn \l_@@_split_int
7900 {
7901     \group_begin:
7902     \dim_set:Nn \col@sep
7903     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7904     \pgftransformshift
7905     {
7906         \pgfpoint
7907         {
7908             \l_tmpa_dim + ##1 \l_tmpb_dim -
7909             \str_case:on \l_@@_hpos_block_str
7910             {
7911                 l { \l_tmpb_dim + \col@sep}
7912                 c { 0.5 \l_tmpb_dim }
7913                 r { \col@sep }
7914             }
7915         }
7916         { \l_@@_tmpc_dim }
7917     }
7918     \pgfset { inner~sep = \c_zero_dim }
7919     \pgfnode
7920     { rectangle }
7921     {
7922         \str_case:on \l_@@_hpos_block_str
7923         {
7924             c { base }
7925             l { base~west }
7926             r { base~east }
7927         }
7928     }
7929     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7930     \group_end:
7931 }
7932 \endpgfpicture
7933 }

```

Now the case where there is no ampersand & in the content of the block.

```

7934 {
7935     \bool_if:NTF \l_@@_p_block_bool
7936     {

```

When the final user has used the key p, we have to compute the width.

```

7937     \pgfpicture
7938     \pgfrememberpicturepositiononpagetrue
7939     \pgf@relevantforpicturesizefalse
7940     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7941     {
7942         \@@_qpoint:n { col - #2 }
7943         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7944         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7945     }
7946     {
7947         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7948         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7949         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7950     }
7951     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }

```



```

7952     \endpgfpicture
7953     \hbox_set:Nn \l_@@_cell_box
7954     {
7955         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7956         { \g_tmpb_dim }
7957         \str_case:on \l_@@_hpos_block_str
7958         { c \centering r \raggedleft l \raggedright j { } }
7959         #6
7960         \end { minipage }
7961     }
7962 }
7963 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7964 \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.

```

7965     \pgfpicture
7966     \pgfrememberpicturepositiononpagetrue
7967     \pgf@relevantforpicturesizefalse
7968     \bool_lazy_any:nTF
7969     {
7970         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7971         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7972         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7973         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7974     }
7975     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7976         \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`.

```

7977         \bool_if:nT \g_@@_last_col_found_bool
7978         {
7979             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7980             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7981         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7982         \tl_set:Ne \l_tmpa_tl
7983         {
7984             \str_case:on \l_@@_vpos_block_str
7985             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7986             { } { % added 2024-06-29
7987                 \str_case:on \l_@@_hpos_block_str
7988                 {
7989                     c { center }
7990                     l { west }
7991                     r { east }
7992                     j { center }
7993                 }
7994             }
7995         c {
7996             \str_case:on \l_@@_hpos_block_str
7997             {
7998                 c { center }
7999                 l { west }
8000                 r { east }
8001                 j { center }
8002             }

```

```

8003     }
8004     T {
8005         \str_case:on \l_@@_hpos_block_str
8006         {
8007             c { north }
8008             l { north-west }
8009             r { north-east }
8010             j { north }
8011         }
8012     }
8013
8014     }
8015     B {
8016         \str_case:on \l_@@_hpos_block_str
8017         {
8018             c { south }
8019             l { south-west }
8020             r { south-east }
8021             j { south }
8022         }
8023     }
8024
8025     }
8026 }
8027 \pgftransformshift
8028 {
8029     \pgfpointanchor
8030     {
8031         \@@_env: - #1 - #2 - block
8032         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8033     }
8034     { \l_tmpa_tl }
8035 }
8036 \pgfset { inner~sep = \c_zero_dim }
8037 \pgfnode
8038 { rectangle }
8039 { \l_tmpa_tl }
8040 { \box_use_drop:N \l_@@_cell_box } { } { }
8041 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

8042 {
8043     \pgfextracty \l_tmpa_dim
8044     {
8045         \@@_qpoint:n
8046         {
8047             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8048             - base
8049         }
8050     }
8051     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

8052 \pgfpointanchor
8053 {
8054     \@@_env: - #1 - #2 - block
8055     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8056 }
8057 {
8058     \str_case:on \l_@@_hpos_block_str
8059     {
8060         c { center }
8061         l { west }

```

```

8062         r { east }
8063         j { center }
8064     }
8065 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8066     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8067     \pgfset { inner~sep = \c_zero_dim }
8068     \pgfnode
8069     { rectangle }
8070     {
8071         \str_case:on \l_@@_hpos_block_str
8072         {
8073             c { base }
8074             l { base~west }
8075             r { base~east }
8076             j { base }
8077         }
8078     }
8079     { \box_use_drop:N \l_@@_cell_box } { } { }
8080 }
8081 \endpgfpicture
8082 }
8083 \group_end:
8084 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8085 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8086 {
8087     \pgfpicture
8088     \pgfrememberpicturepositiononpagetrue
8089     \pgf@relevantforpicturesizefalse
8090     \pgfpathrectanglecorners
8091     { \pgfpoint { #2 } { #3 } }
8092     { \pgfpoint { #4 } { #5 } }
8093     \pgfsetfillcolor { #1 }
8094     \pgfusepath { fill }
8095     \endpgfpicture
8096 }

```

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).

```

8097 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8098 {
8099     \tl_if_empty:NF \l_@@_opacity_tl
8100     {
8101         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8102             {
8103                 \tl_set:Ne \l_@@_fill_tl
8104                 {
8105                     [ opacity = \l_@@_opacity_tl ,
8106                     \tl_tail:o \l_@@_fill_tl
8107                 }
8108             }
8109         {
8110             \tl_set:Ne \l_@@_fill_tl
8111             { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8112         }
8113     }
8114 }

```

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).

```

8115 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8116 {
8117   \group_begin:
8118   \tl_clear:N \l_@@_draw_tl
8119   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8120   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8121   \pgfpicture
8122   \pgfrememberpicturepositiononpagetrue
8123   \pgf@relevantforpicturesizefalse
8124   \tl_if_empty:NF \l_@@_draw_tl
8125   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8126     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8127     { \CT@arc@ }
8128     { \@@_color:o \l_@@_draw_tl }
8129   }
8130   \pgfsetcornersarced
8131   {
8132     \pgfpoint
8133     { \l_@@_rounded_corners_dim }
8134     { \l_@@_rounded_corners_dim }
8135   }
8136   \@@_cut_on_hyphen:w #2 \q_stop
8137   \int_compare:nNnF \l_tmpa_tl > \c@iRow
8138   {
8139     \int_compare:nNnF \l_tmpb_tl > \c@jCol
8140     {
8141       \@@_qpoint:n { row - \l_tmpa_tl }
8142       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8143       \@@_qpoint:n { col - \l_tmpb_tl }
8144       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8145       \@@_cut_on_hyphen:w #3 \q_stop
8146       \int_compare:nNnT \l_tmpa_tl > \c@iRow
8147       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8148       \int_compare:nNnT \l_tmpb_tl > \c@jCol
8149       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8150       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8151       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8152       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8153       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8154       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8155       \pgfpathrectanglecorners
8156       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8157       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8158       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8159       { \pgfusepathqstroke }
8160       { \pgfusepath { stroke } }
8161     }
8162   }
8163   \endpgfpicture
8164   \group_end:
8165 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8166 \keys_define:nn { nicematrix / BlockStroke }
8167 {
8168   color .tl_set:N = \l_@@_draw_tl ,
8169   draw .code:n =
8170     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,

```

```

8171     draw .default:n = default ,
8172     line-width .dim_set:N = \l_@@_line_width_dim ,
8173     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8174     rounded-corners .default:n = 4 pt
8175 }

```

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).

```

8176 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8177 {
8178   \group_begin:
8179   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8180   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8181   \@@_cut_on_hyphen:w #2 \q_stop
8182   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8183   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8184   \@@_cut_on_hyphen:w #3 \q_stop
8185   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8186   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8187   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8188   {
8189     \use:e
8190     {
8191       \@@_vline:n
8192       {
8193         position = ##1 ,
8194         start = \l_@@_tmpc_tl ,
8195         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8196         total-width = \dim_use:N \l_@@_line_width_dim
8197       }
8198     }
8199   }
8200   \group_end:
8201 }

8202 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8203 {
8204   \group_begin:
8205   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8206   \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
8207   \@@_cut_on_hyphen:w #2 \q_stop
8208   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8209   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8210   \@@_cut_on_hyphen:w #3 \q_stop
8211   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8212   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8213   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8214   {
8215     \use:e
8216     {
8217       \@@_hline:n
8218       {
8219         position = ##1 ,
8220         start = \l_@@_tmpd_tl ,
8221         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8222         total-width = \dim_use:N \l_@@_line_width_dim
8223       }
8224     }
8225   }
8226   \group_end:
8227 }

```

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).

```

8228 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8229 {
8230   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8231   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8232   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8233     { \@@_error:n { borders~forbidden } }
8234     {
8235       \tl_clear_new:N \l_@@_borders_tikz_tl
8236       \keys_set:no
8237         { nicematrix / OnlyForTikzInBorders }
8238         \l_@@_borders_clist
8239         \@@_cut_on_hyphen:w #2 \q_stop
8240         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8241         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8242         \@@_cut_on_hyphen:w #3 \q_stop
8243         \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8244         \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8245         \@@_stroke_borders_block_i:
8246     }
8247 }

8248 \hook_gput_code:nnn { begindocument } { . }
8249 {
8250   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8251   {
8252     \c_@@_pgfortikzpicture_tl
8253     \@@_stroke_borders_block_ii:
8254     \c_@@_endpgfortikzpicture_tl
8255   }
8256 }

8257 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8258 {
8259   \pgfrememberpicturepositiononpagetrue
8260   \pgf@relevantforpicturesizefalse
8261   \CT@arc@
8262   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8263   \clist_if_in:NnT \l_@@_borders_clist { right }
8264     { \@@_stroke_vertical:n \l_tmpb_tl }
8265   \clist_if_in:NnT \l_@@_borders_clist { left }
8266     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8267   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8268     { \@@_stroke_horizontal:n \l_tmpa_tl }
8269   \clist_if_in:NnT \l_@@_borders_clist { top }
8270     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8271 }

8272 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8273 {
8274   tikz .code:n =
8275     \cs_if_exist:NTF \tikzpicture
8276       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8277       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8278   tikz .value_required:n = true ,
8279   top .code:n = ,
8280   bottom .code:n = ,
8281   left .code:n = ,
8282   right .code:n = ,
8283   unknown .code:n = \@@_error:n { bad~border }
8284 }

```

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).

```

8285 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8286 {
8287   \@@_qpoint:n \l_@@_tmpc_tl
8288   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8289   \@@_qpoint:n \l_tmpa_tl
8290   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8291   \@@_qpoint:n { #1 }
8292   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8293   {
8294     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8295     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8296     \pgfusepathqstroke
8297   }
8298   {
8299     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8300       ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8301   }
8302 }

```

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).

```

8303 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8304 {
8305   \@@_qpoint:n \l_@@_tmpd_tl
8306   \clist_if_in:NnTF \l_@@_borders_clist { left }
8307     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8308     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8309   \@@_qpoint:n \l_tmpb_tl
8310   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8311   \@@_qpoint:n { #1 }
8312   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8313   {
8314     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8315     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8316     \pgfusepathqstroke
8317   }
8318   {
8319     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8320       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8321   }
8322 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8323 \keys_define:nn { nicematrix / BlockBorders }
8324 {
8325   borders .clist_set:N = \l_@@_borders_clist ,
8326   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8327   rounded-corners .default:n = 4 pt ,
8328   line-width .dim_set:N = \l_@@_line_width_dim
8329 }

```

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.

```

8330 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8331 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5

```

```

8332 {
8333   \begin { tikzpicture }
8334   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8335   \clist_map_inline:nn { #1 }
8336   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8337     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8338     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8339     (
8340       [
8341         xshift = \dim_use:N \l_@@_offset_dim ,
8342         yshift = - \dim_use:N \l_@@_offset_dim
8343       ]
8344       #2 -| #3
8345     )
8346     rectangle
8347     (
8348       [
8349         xshift = - \dim_use:N \l_@@_offset_dim ,
8350         yshift = \dim_use:N \l_@@_offset_dim
8351       ]
8352       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8353     ) ;
8354   }
8355 \end { tikzpicture }
8356 }

```

```

8357 \keys_define:nn { nicematrix / SpecialOffset }
8358 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock:` which has the same syntax as the standard command `\Block` but which is no-op.

```

8359 \cs_new_protected:Npn \@@_NullBlock:
8360 { \@@_collect_options:n { \@@_NullBlock_i: } }
8361 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8362 { }

```

27 How to draw the dotted lines transparently

```

8363 \cs_set_protected:Npn \@@_renew_matrix:
8364 {
8365   \RenewDocumentEnvironment { pmatrix } { } {
8366     { \pNiceMatrix }
8367     { \endpNiceMatrix }
8368   \RenewDocumentEnvironment { vmatrix } { } {
8369     { \vNiceMatrix }
8370     { \endvNiceMatrix }
8371   \RenewDocumentEnvironment { Vmatrix } { } {
8372     { \VNiceMatrix }
8373     { \endVNiceMatrix }
8374   \RenewDocumentEnvironment { bmatrix } { } {
8375     { \bNiceMatrix }
8376     { \endbNiceMatrix }
8377   \RenewDocumentEnvironment { Bmatrix } { } {
8378     { \BNiceMatrix }
8379     { \endBNiceMatrix }

```



```
8380 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
8381 \keys_define:nn { nicematrix / Auto }
8382 {
8383   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8384   columns-type .value_required:n = true ,
8385   l .meta:n = { columns-type = l } ,
8386   r .meta:n = { columns-type = r } ,
8387   c .meta:n = { columns-type = c } ,
8388   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8389   delimiters / color .value_required:n = true ,
8390   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8391   delimiters / max-width .default:n = true ,
8392   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8393   delimiters .value_required:n = true ,
8394   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8395   rounded-corners .default:n = 4 pt
8396 }

8397 \NewDocumentCommand \AutoNiceMatrixWithDelims
8398 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8399 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8400 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8401 {
```

The group is for the protection of the keys.

```
8402   \group_begin:
8403   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8404   \use:e
8405   {
8406     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8407     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8408     [ \exp_not:o \l_tmpa_tl ]
8409   }
8410   \int_if_zero:nT \l_@@_first_row_int
8411   {
8412     \int_if_zero:nT \l_@@_first_col_int { & }
8413     \prg_replicate:nn { #4 - 1 } { & }
8414     \int_compare:nNt \l_@@_last_col_int > { -1 } { & } \\
8415   }
8416   \prg_replicate:nn { #3 }
8417   {
8418     \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`).

```
8419     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8420     \int_compare:nNt \l_@@_last_col_int > { -1 } { & } \\
8421   }
8422   \int_compare:nNt \l_@@_last_row_int > { -2 }
8423   {
8424     \int_if_zero:nT \l_@@_first_col_int { & }
8425     \prg_replicate:nn { #4 - 1 } { & }
8426     \int_compare:nNt \l_@@_last_col_int > { -1 } { & } \\
8427   }
8428   \end { NiceArrayWithDelims }
8429   \group_end:
8430 }
```

```

8431 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8432 {
8433   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8434   {
8435     \bool_gset_true:N \g_@@_delims_bool
8436     \str_gset:Nc \g_@@_name_env_str { #1 AutoNiceMatrix }
8437     \AutoNiceMatrixWithDelims { #2 } { #3 }
8438   }
8439 }
8440 \@@_define_com:nnn p ( )
8441 \@@_define_com:nnn b [ ]
8442 \@@_define_com:nnn v | |
8443 \@@_define_com:nnn V \ | \ |
8444 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8445 \NewDocumentCommand \AutoNiceMatrix { O { } } m O { } m ! O { } }
8446 {
8447   \group_begin:
8448   \bool_gset_false:N \g_@@_delims_bool
8449   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8450   \group_end:
8451 }

```

29 The redefinition of the command `\dotfill`

```

8452 \cs_set_eq:NN \@@_old_dotfill \dotfill
8453 \cs_new_protected:Npn \@@_dotfill:
8454 {

```

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}`).

```

8455   \@@_old_dotfill
8456   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8457 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8458 \cs_new_protected:Npn \@@_dotfill_i:
8459 { \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8460 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8461 {
8462   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8463   {
8464     \@@_actually_diagbox:nnnnn
8465     { \int_use:N \c@iRow }
8466     { \int_use:N \c@jCol }
8467     { \int_use:N \c@iRow }
8468     { \int_use:N \c@jCol }

```

`\g_@@_row_style_tl` contains several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunk of instructions.

```
8469         { \g_@@_row_style_tl \exp_not:n { #1 } }
8470         { \g_@@_row_style_tl \exp_not:n { #2 } }
8471     }
```

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`.

```
8472     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8473     {
8474         { \int_use:N \c@iRow }
8475         { \int_use:N \c@jCol }
8476         { \int_use:N \c@iRow }
8477         { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8478         { }
8479     }
8480 }
```

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.

```
8481 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8482 {
8483     \pgfpicture
8484     \pgf@relevantforpicturesizefalse
8485     \pgfrememberpicturepositiononpagetrue
8486     \@@_qpoint:n { row - #1 }
8487     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8488     \@@_qpoint:n { col - #2 }
8489     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8490     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8491     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8492     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8493     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8494     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8495     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8496 }
```

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.

```
8497     \CT@arc@
8498     \pgfsetroundcap
8499     \pgfusepathqstroke
8500 }
8501 \pgfset { inner~sep = 1 pt }
8502 \pgfscope
8503 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8504 \pgfnode { rectangle } { south~west }
8505 {
8506     \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```
8507     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8508     \end { minipage }
8509 }
8510 { }
8511 { }
```

```

8512 \endpgfscope
8513 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8514 \pgfnode { rectangle } { north~east }
8515 {
8516   \begin { minipage } { 20 cm }
8517   \raggedleft
8518   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8519   \end { minipage }
8520 }
8521 { }
8522 { }
8523 \endpgfpicture
8524 }

```

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. 84.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8525 \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 `\`.

```

8526 \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`.

```

8527 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8528 {
8529   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8530   \@@_CodeAfter_iv:n
8531 }

```

We catch the argument of the command `\end` (in `#1`).

```

8532 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8533 {

```

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.

```

8534   \str_if_eq:eeTF \@currenenvir { #1 }
8535   { \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`.

```

8536   {
8537     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8538     \@@_CodeAfter_ii:n
8539   }
8540 }

```

32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
8541 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8542   {
8543     \pgfpicture
8544     \pgfrememberpicturepositiononpagetrue
8545     \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.

```
8546     \@@_qpoint:n { row - 1 }
8547     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8548     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8549     \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).

```
8550     \bool_if:nTF { #3 }
8551       { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8552       { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8553     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8554       {
8555         \cs_if_exist:cT
8556           { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8557           {
8558             \pgfpointanchor
8559               { \@@_env: - ##1 - #2 }
8560               { \bool_if:nTF { #3 } { west } { east } }
8561             \dim_set:Nn \l_tmpa_dim
8562               { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8563           }
8564       }
```

Now we can put the delimiter with a node of PGF.

```
8565     \pgfset { inner~sep = \c_zero_dim }
8566     \dim_zero:N \nulldelimiterspace
8567     \pgftransformshift
8568       {
8569         \pgfpoint
8570           { \l_tmpa_dim }
8571           { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8572       }
8573     \pgfnode
8574       { rectangle }
8575       { \bool_if:nTF { #3 } { east } { west } }
8576       {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
8577     \nullfont
8578     \c_math_toggle_token
8579     \@@_color:o \l_@@_delimiters_color_tl
8580     \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

8581     \vcenter
8582     {
8583         \nullfont
8584         \hrule \@height
8585             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8586             \@depth \c_zero_dim
8587             \@width \c_zero_dim
8588     }
8589     \bool_if:nTF { #3 } { \right . } { \right #1 }
8590     \c_math_toggle_token
8591 }
8592 { }
8593 { }
8594 \endpgfpicture
8595 }

```

33 The command `\SubMatrix`

```

8596 \keys_define:nn { nicematrix / sub-matrix }
8597 {
8598     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8599     extra-height .value_required:n = true ,
8600     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8601     left-xshift .value_required:n = true ,
8602     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8603     right-xshift .value_required:n = true ,
8604     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8605     xshift .value_required:n = true ,
8606     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8607     delimiters / color .value_required:n = true ,
8608     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8609     slim .default:n = true ,
8610     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8611     hlines .default:n = all ,
8612     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8613     vlines .default:n = all ,
8614     hvlines .meta:n = { hlines, vlines } ,
8615     hvlines .value_forbidden:n = true
8616 }
8617 \keys_define:nn { nicematrix }
8618 {
8619     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8620     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8621     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8622     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8623 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8624 \keys_define:nn { nicematrix / SubMatrix }
8625 {
8626     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8627     delimiters / color .value_required:n = true ,
8628     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8629     hlines .default:n = all ,
8630     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8631     vlines .default:n = all ,
8632     hvlines .meta:n = { hlines, vlines } ,
8633     hvlines .value_forbidden:n = true ,
8634     name .code:n =

```

```

8635 \tl_if_empty:nTF { #1 }
8636 { \@@_error:n { Invalid-name } }
8637 {
8638   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8639   {
8640     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8641     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8642     {
8643       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8644       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8645     }
8646   }
8647   { \@@_error:n { Invalid-name } }
8648 } ,
8649 name .value_required:n = true ,
8650 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8651 rules .value_required:n = true ,
8652 code .tl_set:N = \l_@@_code_tl ,
8653 code .value_required:n = true ,
8654 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8655 }

8656 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8657 {
8658   \peek_remove_spaces:n
8659   {
8660     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8661     {
8662       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8663       [
8664         delimiters / color = \l_@@_delimiters_color_tl ,
8665         hlines = \l_@@_submatrix_hlines_clist ,
8666         vlines = \l_@@_submatrix_vlines_clist ,
8667         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8668         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8669         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8670         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8671         #5
8672       ]
8673     }
8674     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8675   }
8676 }

8677 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8678 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8679 { \@@_SubMatrix_in_code_before_i:n n n n #1 #2 }

8680 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:n n n n #1 #2 #3 #4
8681 {
8682   \seq_gput_right:Ne \g_@@_submatrix_seq
8683   {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8684     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8685     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8686     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8687     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8688   }
8689 }

```

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`.

```

8690 \hook_gput_code:nnn { begindocument } { . }
8691 {
8692   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8693   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8694   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8695     {
8696       \peek_remove_spaces:n
8697         {
8698           \@@_sub_matrix:nnnnnnn
8699             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8700         }
8701     }
8702 }

```

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).

```

8703 \NewDocumentCommand \@@_compute_i_j:nn
8704 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8705 { \@@_compute_i_j:nnnn #1 #2 }
8706 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8707 {
8708   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8709   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8710   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8711   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8712   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8713     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8714   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8715     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8716   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8717     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8718   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8719     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8720 }
8721 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8722 {
8723   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8724 \@@_compute_i_j:nn { #2 } { #3 }
8725 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8726   { \cs_set_nopar:Npn \arraystretch { 1 } }
8727 \bool_lazy_or:nnTF
8728   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8729   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8730   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8731   {
8732     \str_clear_new:N \l_@@_submatrix_name_str
8733     \keys_set:nn { nicematrix / SubMatrix } { #5 }

```



```

8734 \pgfpicture
8735 \pgfrememberpicturepositiononpagetrue
8736 \pgf@relevantforpicturesizefalse
8737 \pgfset { inner~sep = \c_zero_dim }
8738 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8739 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8740 \bool_if:NTF \l_@@_submatrix_slim_bool
8741 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8742 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8743 {
8744   \cs_if_exist:cT
8745   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8746   {
8747     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8748     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8749     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8750   }
8751   \cs_if_exist:cT
8752   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8753   {
8754     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8755     \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8756     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8757   }
8758 }
8759 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8760 { \@@_error:nn { Impossible~delimiter } { left } }
8761 {
8762   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8763   { \@@_error:nn { Impossible~delimiter } { right } }
8764   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8765 }
8766 \endpgfpicture
8767 }
8768 \group_end:
8769 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8770 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8771 {
8772   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8773   \dim_set:Nn \l_@@_y_initial_dim
8774   {
8775     \fp_to_dim:n
8776     {
8777       \pgf@y
8778       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8779     }
8780   }
8781   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8782   \dim_set:Nn \l_@@_y_final_dim
8783   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8784   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8785   {
8786     \cs_if_exist:cT
8787     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8788     {
8789       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8790       \dim_set:Nn \l_@@_y_initial_dim
8791       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8792     }

```

```

8793     \cs_if_exist:cT
8794     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8795     {
8796         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8797         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8798         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8799     }
8800 }
8801 \dim_set:Nn \l_tmpa_dim
8802 {
8803     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8804     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8805 }
8806 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8807     \group_begin:
8808     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8809     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8810     \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`.

```

8811     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8812     {
8813         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8814         {
8815             \int_compare:nNnT
8816             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8817             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8818                 \@@_qpoint:n { col - ##1 }
8819                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8820                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8821                 \pgfusepathqstroke
8822             }
8823         }
8824     }

```

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.

```

8825     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8826     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8827     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8828     {
8829         \bool_lazy_and:nnTF
8830         { \int_compare_p:nNn { ##1 } > \c_zero_int }
8831         {
8832             \int_compare_p:nNn
8833             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8834         {
8835             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8836             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8837             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8838             \pgfusepathqstroke
8839         }
8840         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8841     }

```

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.

```

8842 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8843 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8844 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8845 {
8846   \bool_lazy_and:nnTF
8847   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8848   {
8849     \int_compare_p:nNn
8850     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8851   {
8852     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
8853   \group_begin:
```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8854   \dim_set:Nn \l_tmpa_dim
8855   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8856   \str_case:nn { #1 }
8857   {
8858     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8859     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8860     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } } }
8861   }
8862   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8863   \dim_set:Nn \l_tmpb_dim
8864   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8865   \str_case:nn { #2 }
8866   {
8867     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8868     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8869     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8870   }
8871   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8872   \pgfusepathqstroke
8873   \group_end:
8874 }
8875 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8876 }

```

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).

```

8877 \str_if_empty:NF \l_@@_submatrix_name_str
8878 {
8879   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8880   \l_@@_x_initial_dim \l_@@_y_initial_dim
8881   \l_@@_x_final_dim \l_@@_y_final_dim
8882 }
8883 \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`.

```

8884 \begin { pgfscope }
8885 \pgftransformshift
8886 {
8887   \pgfpoint
8888   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8889   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8890 }
8891 \str_if_empty:NTF \l_@@_submatrix_name_str
8892 { \@@_node_left:nn #1 { } }

```

```

8893     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8894 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8895 \pgftransformshift
8896 {
8897   \pgfpoint
8898   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8899   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8900 }
8901 \str_if_empty:NTF \l_@@_submatrix_name_str
8902 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8903 {
8904   \@@_node_right:nnnn #2
8905   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8906 }

```

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`.

```

8907 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8908 \flag_clear_new:N \l_@@_code_flag
8909 \l_@@_code_tl
8910 }

```

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 $i-lj$ 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`.

```

8911 \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 currying.

```

8912 \cs_new:Npn \@@_pgfpointanchor:n #1
8913 { \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`).

```

8914 \cs_new:Npn \@@_pgfpointanchor_i:n #1
8915 { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
8916 \cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
8917 {

```

The command `\str_if_empty:nTF` is "fully expandable".

```

8918 \str_if_empty:nTF { #1 }

```

First, when the name of the name begins with `\tikz@pp@name`.

```

8919 { \@@_pgfpointanchor_iv:w #2 }

```

And now, when there is no `\tikz@pp@name`.

```

8920 { \@@_pgfpointanchor_ii:n { #1 } }
8921 }

```

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`).

```
8922 \cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
8923 { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` or of the form `i-j` (with an hyphen).

Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package `etl` but, as of now, we do not load `etl`.

```
8924 \cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1-\q_stop }
```

```
8925 \cs_new:Npn \@@_pgfpointanchor_i:w #1-#2\q_stop
8926 {
```

The command `\str_if_empty:nTF` is “fully expandable”.

```
8927 \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
8928 { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retrieve the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
8929 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8930 }
```

The following function is for the case when the name contains an hyphen.

```
8931 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8932 {
```

We have to add the prefix `\@@_env:` “by hand” since we have retrieved the potential `\tikz@pp@name`.

```
8933 \@@_env:
8934 - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8935 - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8936 }
```

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.

```
8937 \tl_const:Nn \c_@@_integers_alist_tl
8938 {
8939 { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8940 { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8941 { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8942 { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8943 }
```

```
8944 \cs_new:Npn \@@_pgfpointanchor_iii:n #1
8945 {
```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form `i-j`. That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises with its command `\name_of_command` (see above).

In that case, the `i` of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the `j` arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```
8946 \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8947 {
8948 \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env`: “by hand” since we have retrieved the potential `\tikz@pp@name`.

```

8949     \@@_env: -
8950     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8951       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8952       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8953   }
8954   {
8955     \str_if_eq:eeTF { #1 } { last }
8956     {
8957       \flag_raise:N \l_@@_code_flag
8958       \@@_env: -
8959       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8960         { \int_eval:n { \l_@@_last_i_tl + 1 } }
8961         { \int_eval:n { \l_@@_last_j_tl + 1 } }
8962     }
8963     { #1 }
8964   }
8965 }

```

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`).

```

8966 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8967 {
8968   \pgfnode
8969     { rectangle }
8970     { east }
8971     {
8972       \nullfont
8973       \c_math_toggle_token
8974       \@@_color:o \l_@@_delimiters_color_tl
8975       \left #1
8976       \vcenter
8977         {
8978           \nullfont
8979           \hrule \@height \l_tmpa_dim
8980             \@depth \c_zero_dim
8981             \@width \c_zero_dim
8982         }
8983       \right .
8984       \c_math_toggle_token
8985     }
8986     { #2 }
8987     { }
8988 }

```

The command `\@@_node_right:nnn` 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.

```

8989 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8990 {
8991   \pgfnode
8992     { rectangle }
8993     { west }
8994     {
8995       \nullfont
8996       \c_math_toggle_token
8997       \colorlet { current-color } { . }
8998       \@@_color:o \l_@@_delimiters_color_tl
8999       \left .
9000       \vcenter
9001         {

```

```

9002         \nullfont
9003         \hrule \@height \l_tmpa_dim
9004             \@depth \c_zero_dim
9005             \@width \c_zero_dim
9006     }
9007     \right #1
9008     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
9009     ^ { \color { current-color } \smash { #4 } }
9010     \c_math_toggle_token
9011 }
9012 { #2 }
9013 { }
9014 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

9015 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
9016 {
9017   \peek_remove_spaces:n
9018   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
9019 }
9020 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
9021 {
9022   \peek_remove_spaces:n
9023   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
9024 }
9025 \keys_define:nn { nicematrix / Brace }
9026 {
9027   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
9028   left-shorten .default:n = true ,
9029   left-shorten .value_forbidden:n = true ,
9030   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
9031   right-shorten .default:n = true ,
9032   right-shorten .value_forbidden:n = true ,
9033   shorten .meta:n = { left-shorten , right-shorten } ,
9034   shorten .value_forbidden:n = true ,
9035   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
9036   yshift .value_required:n = true ,
9037   yshift .initial:n = \c_zero_dim ,
9038   color .tl_set:N = \l_tmpa_tl ,
9039   color .value_required:n = true ,
9040   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9041 }

```

`#1` is the first cell of the rectangle (with the syntax `i-lj`; `#2` is the last cell of the rectangle; `#3` is the label of the text; `#4` is the optional argument (a list of *key-value* pairs); `#5` is equal to `under` or `over`.

```

9042 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9043 {
9044   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9045   \@@_compute_i_j:nn { #1 } { #2 }
9046   \bool_lazy_or:nnTF
9047     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9048     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }

```

```

9049 {
9050   \str_if_eq:eeTF { #5 } { under }
9051   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9052   { \@@_error:nn { Construct-too-large } { \OverBrace } }
9053 }
9054 {
9055   \tl_clear:N \l_tmpa_tl
9056   \keys_set:nn { nicematrix / Brace } { #4 }
9057   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9058   \pgfpicture
9059   \pgfrememberpicturepositiononpagetrue
9060   \pgf@relevantforpicturesizefalse
9061   \bool_if:NT \l_@@_brace_left_shorten_bool
9062   {
9063     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9064     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9065     {
9066       \cs_if_exist:cT
9067       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9068       {
9069         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9070
9071         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9072         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9073       }
9074     }
9075   }
9076   \bool_lazy_or:nnT
9077   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9078   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9079   {
9080     \@@_qpoint:n { col - \l_@@_first_j_tl }
9081     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9082   }
9083   \bool_if:NT \l_@@_brace_right_shorten_bool
9084   {
9085     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9086     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9087     {
9088       \cs_if_exist:cT
9089       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9090       {
9091         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9092         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9093         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9094       }
9095     }
9096   }
9097   \bool_lazy_or:nnT
9098   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9099   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9100   {
9101     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9102     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9103   }
9104   \pgfset { inner~sep = \c_zero_dim }
9105   \str_if_eq:eeTF { #5 } { under }
9106   { \@@_underbrace_i:n { #3 } }
9107   { \@@_overbrace_i:n { #3 } }
9108   \endpgfpicture
9109 }
9110 \group_end:
9111 }

```


The argument is the text to put above the brace.

```

9112 \cs_new_protected:Npn \@@_overbrace_i:n #1
9113 {
9114   \@@_qpoint:n { row - \l_@@_first_i_tl }
9115   \pgftransformshift
9116     {
9117       \pgfpoint
9118         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
9119         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
9120     }
9121   \pgfnode
9122     { rectangle }
9123     { south }
9124     {
9125       \vtop
9126         {
9127           \group_begin:
9128           \everycr { }
9129           \halign
9130             {
9131               \hfil ## \hfil \crcr
9132               \bool_if:NTF \l_@@_tabular_bool
9133                 { \begin { tabular } { c } #1 \end { tabular } }
9134                 { $ \begin { array } { c } #1 \end { array } $ }
9135               \cr
9136               \c_math_toggle_token
9137               \overbrace
9138                 {
9139                   \hbox_to_wd:nn
9140                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9141                     { }
9142                 }
9143               \c_math_toggle_token
9144             \cr
9145           }
9146         \group_end:
9147       }
9148     }
9149   { }
9150 }
9151 }

```

The argument is the text to put under the brace.

```

9152 \cs_new_protected:Npn \@@_underbrace_i:n #1
9153 {
9154   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9155   \pgftransformshift
9156     {
9157       \pgfpoint
9158         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
9159         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9160     }
9161   \pgfnode
9162     { rectangle }
9163     { north }
9164     {
9165       \group_begin:
9166       \everycr { }
9167       \vbox
9168         {
9169           \halign
9170             {
9171               \hfil ## \hfil \crcr

```

```

9172         \c_math_toggle_token
9173         \underbrace
9174         {
9175             \hbox_to_wd:nn
9176             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9177             { }
9178         }
9179         \c_math_toggle_token
9180         \cr
9181         \bool_if:NTF \l_@@_tabular_bool
9182         { \begin { tabular } { c } #1 \end { tabular } }
9183         { $ \begin { array } { c } #1 \end { array } $ }
9184         \cr
9185     }
9186 }
9187 \group_end:
9188 }
9189 { }
9190 { }
9191 }

```

35 The commands HBrace et VBrace

```

9192 \hook_gput_code:nnn { begindocument } { . }
9193 {
9194     \cs_if_exist:cT { tikz@library@decorations.pathreplacing@loaded }
9195     {
9196         \tikzset
9197         {
9198             nicematrix / brace / .style =
9199             {
9200                 decoration = { brace , raise = -0.15 em } ,
9201                 decorate ,
9202             } ,

```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```

9203         nicematrix / mirrored-brace / .style =
9204         {
9205             nicematrix / brace ,
9206             decoration = mirror ,
9207         }
9208     }
9209 }
9210 }

```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```

9211 \keys_define:nn { nicematrix / Hbrace }
9212 {
9213     color .code:n = ,
9214     horizontal-labels .code:n = ,
9215     shorten .code:n = ,
9216     shorten-start .code:n = ,
9217     shorten-end .code:n = ,
9218     unknown .code:n = \@@_error:n { Unknown~key~for~Hbrace }
9219 }

```

Here we need an “fully expandable” command.

```

9220 \NewExpandableDocumentCommand { \@@_Hbrace } { 0 { } m m }
9221 {
9222   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9223     { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
9224     { \@@_error:n { Hbrace~not~allowed } }
9225 }

```

The following command must *not* be protected.

```

9226 \cs_new:Npn \@@_hbrace:nnn #1 #2 #3
9227 {
9228   \int_compare:nNnTF \c@iRow < 1
9229   {

```

We recall that `\str_if_eq:nnTF` is “fully expandable”.

```

9230   \str_if_eq:nnTF { #2 } { * }
9231   {
9232     \NiceMatrixOptions{nullify-dots}
9233     \Ldots
9234     [
9235       line-style = nicematrix / brace ,
9236       #1 ,
9237       up =
9238         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9239     ]
9240   }
9241   {
9242     \Hdotsfor
9243     [
9244       line-style = nicematrix / brace ,
9245       #1 ,
9246       up =
9247         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9248     ]
9249     { #2 }
9250   }
9251 }
9252 {
9253   \str_if_eq:nnTF { #2 } { * }
9254   {
9255     \NiceMatrixOptions{nullify-dots}
9256     \Ldots
9257     [
9258       line-style = nicematrix / mirrored-brace ,
9259       #1 ,
9260       down =
9261         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9262     ]
9263   }
9264   {
9265     \Hdotsfor
9266     [
9267       line-style = nicematrix / mirrored-brace ,
9268       #1 ,
9269       down =
9270         \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9271     ]
9272     { #2 }
9273   }
9274 }
9275 \keys_set:nn { nicematrix / Hbrace } { #1 }
9276 }

```

Here we need an “fully expandable” command.

```

9277 \NewExpandableDocumentCommand { \@@_Vbrace } { 0 { } m m }
9278 {
9279   \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
9280     { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
9281     { \@@_error:n { Vbrace~not~allowed } }
9282 }

```

The following command must *not* be protected.

```

9283 \cs_new:Npn \@@_vbrace:nnn #1 #2 #3
9284 {
9285   \int_compare:nNnTF \c@jCol = 0
9286     {
9287       \str_if_eq:nnTF { #2 } { * }
9288         {
9289           \NiceMatrixOptions{nullify-dots}
9290           \Vdots
9291           [
9292             line-style = nicematrix / mirrored-brace ,
9293             #1 ,
9294             down =
9295               \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9296           ]
9297         }
9298         {
9299           \Vdotsfor
9300           [
9301             line-style = nicematrix / mirrored-brace ,
9302             #1 ,
9303             down =
9304               \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9305           ]
9306           { #2 }
9307         }
9308       }
9309       {
9310         \str_if_eq:nnTF { #2 } { * }
9311           {
9312             \NiceMatrixOptions{nullify-dots}
9313             \Vdots
9314             [
9315               line-style = nicematrix / brace ,
9316               #1 ,
9317               up =
9318                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9319             ]
9320           }
9321           {
9322             \Vdotsfor
9323             [
9324               line-style = nicematrix / brace ,
9325               #1 ,
9326               up =
9327                 \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
9328             ]
9329             { #2 }
9330           }
9331         }
9332       \keys_set:nn { nicematrix / Hbrace } { #1 }
9333     }

```

36 The command TikzEveryCell

```

9334 \bool_new:N \l_@@_not_empty_bool
9335 \bool_new:N \l_@@_empty_bool
9336
9337 \keys_define:nn { nicematrix / TikzEveryCell }
9338 {
9339   not-empty .code:n =
9340     \bool_lazy_or:nnTF
9341       \l_@@_in_code_after_bool
9342       \g_@@_recreate_cell_nodes_bool
9343       { \bool_set_true:N \l_@@_not_empty_bool }
9344       { \@@_error:n { detection~of~empty~cells } } ,
9345   not-empty .value_forbidden:n = true ,
9346   empty .code:n =
9347     \bool_lazy_or:nnTF
9348       \l_@@_in_code_after_bool
9349       \g_@@_recreate_cell_nodes_bool
9350       { \bool_set_true:N \l_@@_empty_bool }
9351       { \@@_error:n { detection~of~empty~cells } } ,
9352   empty .value_forbidden:n = true ,
9353   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9354 }
9355
9356
9357 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9358 {
9359   \IfPackageLoadedTF { tikz }
9360   {
9361     \group_begin:
9362     \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.

```

9363     \tl_set:Nn \l_tmpa_tl { { #2 } }
9364     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9365       { \@@_for_a_block:nnnnn ##1 }
9366     \@@_all_the_cells:
9367     \group_end:
9368   }
9369   { \@@_error:n { TikzEveryCell~without~tikz } }
9370 }
9371
9372 \tl_new:N \@@_i_tl
9373 \tl_new:N \@@_j_tl
9374
9375
9376 \cs_new_protected:Nn \@@_all_the_cells:
9377 {
9378   \int_step_variable:nNn \c@iRow \@@_i_tl
9379   {
9380     \int_step_variable:nNn \c@jCol \@@_j_tl
9381     {
9382       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9383       {
9384         \clist_if_in:NeF \l_@@_corners_cells_clist
9385           { \@@_i_tl - \@@_j_tl }
9386         {
9387           \bool_set_false:N \l_tmpa_bool
9388           \cs_if_exist:cTF
9389             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9390             {
9391               \bool_if:NF \l_@@_empty_bool
9392               { \bool_set_true:N \l_tmpa_bool }
9393             }
9394         }

```

```

9395             \bool_if:NF \l_@@_not_empty_bool
9396             { \bool_set_true:N \l_tmpa_bool }
9397         }
9398     \bool_if:NT \l_tmpa_bool
9399     {
9400         \@@_block_tikz:onnnn
9401         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9402     }
9403 }
9404 }
9405 }
9406 }
9407 }
9408
9409 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9410 {
9411     \bool_if:NF \l_@@_empty_bool
9412     {
9413         \@@_block_tikz:onnnn
9414         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9415     }
9416     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9417 }
9418
9419 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9420 {
9421     \int_step_inline:nnn { #1 } { #3 }
9422     {
9423         \int_step_inline:nnn { #2 } { #4 }
9424         { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9425     }
9426 }

```

37 The command `\ShowCellNames`

```

9427 \NewDocumentCommand \@@_ShowCellNames { }
9428 {
9429     \bool_if:NT \l_@@_in_code_after_bool
9430     {
9431         \pgfpicture
9432         \pgfrememberpicturepositiononpagetrue
9433         \pgf@relevantforpicturesizefalse
9434         \pgfpathrectanglecorners
9435         { \@@_qpoint:n { 1 } }
9436         {
9437             \@@_qpoint:n
9438             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9439         }
9440         \pgfsetfillopacity { 0.75 }
9441         \pgfsetfillcolor { white }
9442         \pgfusepathqfill
9443         \endpgfpicture
9444     }
9445     \dim_gzero_new:N \g_@@_tmpc_dim
9446     \dim_gzero_new:N \g_@@_tmpd_dim
9447     \dim_gzero_new:N \g_@@_tmpe_dim
9448     \int_step_inline:nn \c@iRow
9449     {
9450         \bool_if:NTF \l_@@_in_code_after_bool
9451         {
9452             \pgfpicture
9453             \pgfrememberpicturepositiononpagetrue

```

```

9454     \pgf@relevantforpicturesizefalse
9455   }
9456   { \begin { pgfpicture } }
9457   \@@_qpoint:n { row - ##1 }
9458   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9459   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9460   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9461   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9462   \bool_if:NTF \l_@@_in_code_after_bool
9463     { \endpgfpicture }
9464     { \end { pgfpicture } }
9465   \int_step_inline:nn \c@jCol
9466   {
9467     \hbox_set:Nn \l_tmpa_box
9468     {
9469       \normalfont \Large \sffamily \bfseries
9470       \bool_if:NTF \l_@@_in_code_after_bool
9471         { \color { red } }
9472         { \color { red ! 50 } }
9473       ##1 - ####1
9474     }
9475     \bool_if:NTF \l_@@_in_code_after_bool
9476     {
9477       \pgfpicture
9478       \pgfrememberpicturerepositiononpagetrue
9479       \pgf@relevantforpicturesizefalse
9480     }
9481     { \begin { pgfpicture } }
9482     \@@_qpoint:n { col - ####1 }
9483     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9484     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9485     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9486     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9487     \bool_if:NTF \l_@@_in_code_after_bool
9488       { \endpgfpicture }
9489       { \end { pgfpicture } }
9490     \fp_set:Nn \l_tmpa_fp
9491     {
9492       \fp_min:nn
9493       {
9494         \fp_min:nn
9495         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9496         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9497       }
9498       { 1.0 }
9499     }
9500     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9501     \pgfpicture
9502     \pgfrememberpicturerepositiononpagetrue
9503     \pgf@relevantforpicturesizefalse
9504     \pgftransformshift
9505     {
9506       \pgfpoint
9507       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9508       { \dim_use:N \g_tmpa_dim }
9509     }
9510     \pgfnode
9511     { rectangle }
9512     { center }
9513     { \box_use:N \l_tmpa_box }
9514     { }
9515     { }
9516   \endpgfpicture

```

```

9517     }
9518   }
9519 }

```

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.

```

9520 \bool_new:N \g_@@_footnotehyper_bool

```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

```

9521 \bool_new:N \g_@@_footnote_bool

9522 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9523 {
9524   You-have-used-the-key~'\l_keys_key_str'~when~loading~nicematrix~
9525   but~that~key~is~unknown. \\
9526   It~will~be~ignored. \\
9527   For~a~list~of~the~available~keys,~type~H~<return>.
9528 }
9529 {
9530   The~available~keys~are~(in~alphabetic~order):~
9531   footnote,~
9532   footnotehyper,~
9533   messages-for-Overleaf,~
9534   renew-dots~and~
9535   renew-matrix.
9536 }

9537 \keys_define:nn { nicematrix }
9538 {
9539   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9540   renew-dots .value_forbidden:n = true ,
9541   renew-matrix .code:n = \@@_renew_matrix: ,
9542   renew-matrix .value_forbidden:n = true ,
9543   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9544   footnote .bool_set:N = \g_@@_footnote_bool ,
9545   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9546   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9547 }
9548 \ProcessKeyOptions

9549 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9550 {
9551   You~can't~use~the~option~'footnote'~because~the~package~
9552   footnotehyper~has~already~been~loaded.~
9553   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9554   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9555   of~the~package~footnotehyper.\\
9556   The~package~footnote~won't~be~loaded.
9557 }

9558 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9559 {
9560   You~can't~use~the~option~'footnotehyper'~because~the~package~
9561   footnote~has~already~been~loaded.~

```



```

9562   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9563   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9564   of~the~package~footnote.\\
9565   The~package~footnotehyper~won't~be~loaded.
9566 }

```

```

9567 \bool_if:NT \g_@@_footnote_bool
9568 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9569   \IfClassLoadedTF { beamer }
9570   { \bool_set_false:N \g_@@_footnote_bool }
9571   {
9572     \IfPackageLoadedTF { footnotehyper }
9573     { \@@_error:n { footnote-with-footnotehyper-package } }
9574     { \usepackage { footnote } }
9575   }
9576 }

9577 \bool_if:NT \g_@@_footnotehyper_bool
9578 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9579   \IfClassLoadedTF { beamer }
9580   { \bool_set_false:N \g_@@_footnote_bool }
9581   {
9582     \IfPackageLoadedTF { footnote }
9583     { \@@_error:n { footnotehyper-with-footnote-package } }
9584     { \usepackage { footnotehyper } }
9585   }
9586   \bool_set_true:N \g_@@_footnote_bool
9587 }

```

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.

```

9588 \bool_new:N \l_@@_underscore_loaded_bool
9589 \IfPackageLoadedT { underscore }
9590 { \bool_set_true:N \l_@@_underscore_loaded_bool }

9591 \hook_gput_code:nnn { begindocument } { . }
9592 {
9593   \bool_if:NF \l_@@_underscore_loaded_bool
9594   {
9595     \IfPackageLoadedT { underscore }
9596     { \@@_error:n { underscore-after-nicematrix } }
9597   }
9598 }

```

40 Error messages of the package

```

9599 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9600 { \str_const:Nn \c_@@_available_keys_str { } }
9601 {
9602   \str_const:Nn \c_@@_available_keys_str
9603     { For~a~list~of~the~available~keys,~type~H~<return>. }
9604 }
9605 \seq_new:N \g_@@_types_of_matrix_seq
9606 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9607 {
9608   NiceMatrix ,
9609   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9610 }
9611 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9612 { \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`.

```

9613 \cs_new_protected:Npn \@@_error_too_much_cols:
9614 {
9615   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9616     { \@@_fatal:nn { too-much-cols-for-array } }
9617   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9618     { \@@_fatal:n { too-much-cols-for-matrix } }
9619   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9620     { \@@_fatal:n { too-much-cols-for-matrix } }
9621   \bool_if:NF \l_@@_last_col_without_value_bool
9622     { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9623 }

```

The following command must *not* be protected since it's used in an error message.

```

9624 \cs_new:Npn \@@_message_hdotsfor:
9625 {
9626   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9627     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9628 }
9629 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9630 {
9631   Incompatible~options.\\
9632   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9633   The~output~will~not~be~reliable.
9634 }
9635 \@@_msg_new:nn { key~color~inside }
9636 {
9637   Key~deprecated.\\
9638   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9639   and~have~been~deprecated.\\
9640   You~won't~have~similar~message~till~the~end~of~the~document.
9641 }
9642 \@@_msg_new:nn { negative~weight }
9643 {
9644   Negative~weight.\\
9645   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9646   the~value~'\int_use:N \l_@@_weight_int'.\\
9647   The~absolute~value~will~be~used.
9648 }
9649 \@@_msg_new:nn { last~col~not~used }

```

```

9650 {
9651   Column~not~used.\\
9652   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9653   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9654 }
9655 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9656 {
9657   Too~much~columns.\\
9658   In~the~row~\int_eval:n { \c@iRow },~
9659   you~try~to~use~more~columns~
9660   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\
9661   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9662   (plus~the~exterior~columns).~This~error~is~fatal.
9663 }
9664 \@@_msg_new:nn { too~much~cols~for~matrix }
9665 {
9666   Too~much~columns.\\
9667   In~the~row~\int_eval:n { \c@iRow },~
9668   you~try~to~use~more~columns~than~allowed~by~your~
9669   \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall~that~the~maximal~
9670   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9671   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9672   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9673   \token_to_str:N \setcounter\ to~change~that~value).~
9674   This~error~is~fatal.
9675 }
9676 \@@_msg_new:nn { too~much~cols~for~array }
9677 {
9678   Too~much~columns.\\
9679   In~the~row~\int_eval:n { \c@iRow },~
9680   ~you~try~to~use~more~columns~than~allowed~by~your~
9681   \@@_full_name_env:.~\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9682   \int_use:N \g_@@_static_num_of_col_int\
9683   \bool_if:nT
9684     { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9685     { ~(plus~the~exterior~ones) }
9686   since~the~preamble~is~'\g_@@_user_preamble_tl'.\\
9687   This~error~is~fatal.
9688 }
9689 \@@_msg_new:nn { columns~not~used }
9690 {
9691   Columns~not~used.\\
9692   The~preamble~of~your~\@@_full_name_env:\ is~'\g_@@_user_preamble_tl'.~
9693   It~announces~\int_use:N
9694   \g_@@_static_num_of_col_int\ columns~but~you~only~used~\int_use:N \c@jCol.\\
9695   The~columns~you~did~not~used~won't~be~created.\\
9696   You~won't~have~similar~warning~till~the~end~of~the~document.
9697 }
9698 \@@_msg_new:nn { empty~preamble }
9699 {
9700   Empty~preamble.\\
9701   The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9702   This~error~is~fatal.
9703 }
9704 \@@_msg_new:nn { in~first~col }
9705 {
9706   Erroneous~use.\\
9707   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9708   That~command~will~be~ignored.
9709 }

```

```

9710 \@@_msg_new:nn { in~last~col }
9711 {
9712   Erroneous~use.\\
9713   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9714   That~command~will~be~ignored.
9715 }

9716 \@@_msg_new:nn { in~first~row }
9717 {
9718   Erroneous~use.\\
9719   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9720   That~command~will~be~ignored.
9721 }

9722 \@@_msg_new:nn { in~last~row }
9723 {
9724   Erroneous~use.\\
9725   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9726   That~command~will~be~ignored.
9727 }

9728 \@@_msg_new:nn { TopRule~without~booktabs }
9729 {
9730   Erroneous~use.\\
9731   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9732   That~command~will~be~ignored.
9733 }

9734 \@@_msg_new:nn { TopRule~without~tikz }
9735 {
9736   Erroneous~use.\\
9737   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9738   That~command~will~be~ignored.
9739 }

9740 \@@_msg_new:nn { caption~outside~float }
9741 {
9742   Key~caption~forbidden.\\
9743   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9744   environment.~This~key~will~be~ignored.
9745 }

9746 \@@_msg_new:nn { short~caption~without~caption }
9747 {
9748   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9749   However,~your~'short~caption'~will~be~used~as~'caption'.
9750 }

9751 \@@_msg_new:nn { double~closing~delimiter }
9752 {
9753   Double~delimiter.\\
9754   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9755   delimiter.~This~delimiter~will~be~ignored.
9756 }

9757 \@@_msg_new:nn { delimiter~after~opening }
9758 {
9759   Double~delimiter.\\
9760   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9761   delimiter.~That~delimiter~will~be~ignored.
9762 }

9763 \@@_msg_new:nn { bad~option~for~line~style }
9764 {
9765   Bad~line~style.\\
9766   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9767   is~'standard'.~That~key~will~be~ignored.
9768 }

```

```

9769 \@@_msg_new:nn { corners-with-no-cell-nodes }
9770 {
9771   Incompatible-keys.\
9772   You-can't-use-the-key~'corners'-here-because-the-key~'no-cell-nodes'~
9773   is-in~force.\
9774   If-you-go-on,~that-key-will-be-ignored.
9775 }

9776 \@@_msg_new:nn { extra-nodes-with-no-cell-nodes }
9777 {
9778   Incompatible-keys.\
9779   You-can't-create~'extra-nodes'~here-because-the-key~'no-cell-nodes'~
9780   is-in~force.\
9781   If-you-go-on,~those-extra-nodes-won't-be-created.
9782 }

9783 \@@_msg_new:nn { Identical-notes-in-caption }
9784 {
9785   Identical~tabular~notes.\
9786   You-can't-put~several~notes~with~the~same~content~in~
9787   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
9788   If-you-go-on,~the-output-will~probably~be~erroneous.
9789 }

9790 \@@_msg_new:nn { tabularnote-below-the-tabular }
9791 {
9792   \token_to_str:N \tabularnote\ forbidden\
9793   You-can't-use~\token_to_str:N \tabularnote\ in-the-caption~
9794   of~your~tabular~because~the~caption~will~be~composed~below~
9795   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9796   key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\
9797   Your~\token_to_str:N \tabularnote\ will-be-discarded~and~
9798   no~similar~error~will~raised~in~this~document.
9799 }

9800 \@@_msg_new:nn { Unknown-key-for-rules }
9801 {
9802   Unknown-key.\
9803   There-is-only-two-keys-available-here:~width~and~color.\
9804   Your-key~'\l_keys_key_str'~will-be-ignored.
9805 }

9806 \@@_msg_new:nn { Unknown-key-for-Hbrace }
9807 {
9808   Unknown-key.\
9809   You-have-used~the~key~'\l_keys_key_str'~but~the~only~
9810   keys~allowed~for~the~commands~\token_to_str:N \Hbrace\
9811   and~\token_to_str:N \Vbrace\ are:~'color',~
9812   'horizontal-labels',~'shorten'~'shorten-end'~
9813   and~'shorten-start'.
9814 }

9815 \@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
9816 {
9817   Unknown-key.\
9818   There-is-only-two-keys-available-here:~
9819   'empty'~and~'not-empty'.\
9820   Your-key~'\l_keys_key_str'~will-be-ignored.
9821 }

9822 \@@_msg_new:nn { Unknown-key-for~rotate }
9823 {
9824   Unknown-key.\
9825   The~only~key~available~here~is~'c'.\
9826   Your-key~'\l_keys_key_str'~will-be-ignored.
9827 }

9828 \@@_msg_new:nnn { Unknown-key-for~custom-line }

```

```

9829 {
9830   Unknown~key.\\
9831   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
9832   It~you~go~on,~you~will~probably~have~other~errors. \\
9833   \c_@@_available_keys_str
9834 }
9835 {
9836   The~available~keys~are~(in~alphabetic~order):~
9837   ccommand,~
9838   color,~
9839   command,~
9840   dotted,~
9841   letter,~
9842   multiplicity,~
9843   sep-color,~
9844   tikz,~and~total-width.
9845 }
9846 \@@_msg_new:nnn { Unknown~key~for~xdots }
9847 {
9848   Unknown~key.\\
9849   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9850   \c_@@_available_keys_str
9851 }
9852 {
9853   The~available~keys~are~(in~alphabetic~order):~
9854   'color',~
9855   'horizontal-labels',~
9856   'inter',~
9857   'line-style',~
9858   'radius',~
9859   'shorten',~
9860   'shorten-end'~and~'shorten-start'.
9861 }
9862 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9863 {
9864   Unknown~key.\\
9865   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9866   (and~you~try~to~use~'\l_keys_key_str')\\
9867   That~key~will~be~ignored.
9868 }
9869 \@@_msg_new:nn { label~without~caption }
9870 {
9871   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9872   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9873 }
9874 \@@_msg_new:nn { W~warning }
9875 {
9876   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9877   (row~\int_use:N \c@iRow).
9878 }
9879 \@@_msg_new:nn { Construct~too~large }
9880 {
9881   Construct~too~large.\\
9882   Your~command~\token_to_str:N #1
9883   can't~be~drawn~because~your~matrix~is~too~small.\\
9884   That~command~will~be~ignored.
9885 }
9886 \@@_msg_new:nn { underscore~after~nicematrix }
9887 {
9888   Problem~with~'underscore'.\\
9889   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~

```

```

9890     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9891     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9892 }
9893 \@@_msg_new:nn { ampersand~in~light~syntax }
9894 {
9895     Ampersand~forbidden.\\
9896     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9897     ~the~key~'light~syntax'~is~in~force.~This~error~is~fatal.
9898 }
9899 \@@_msg_new:nn { double~backslash~in~light~syntax }
9900 {
9901     Double~backslash~forbidden.\\
9902     You~can't~use~\token_to_str:N
9903     \\~to~separate~rows~because~the~key~'light~syntax'~
9904     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9905     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9906 }
9907 \@@_msg_new:nn { hlines~with~color }
9908 {
9909     Incompatible~keys.\\
9910     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9911     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9912     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9913     Your~key~will~be~discarded.
9914 }
9915 \@@_msg_new:nn { bad~value~for~baseline }
9916 {
9917     Bad~value~for~baseline.\\
9918     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9919     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9920     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9921     the~form~'line-i'.\\
9922     A~value~of~1~will~be~used.
9923 }
9924 \@@_msg_new:nn { detection~of~empty~cells }
9925 {
9926     Problem~with~'not~empty'\\
9927     For~technical~reasons,~you~must~activate~
9928     'create~cell~nodes'~in~\token_to_str:N \CodeBefore\
9929     in~order~to~use~the~key~'\l_keys_key_str'.\\
9930     That~key~will~be~ignored.
9931 }
9932 \@@_msg_new:nn { siunitx~not~loaded }
9933 {
9934     siunitx~not~loaded\\
9935     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9936     That~error~is~fatal.
9937 }
9938 \@@_msg_new:nn { Invalid~name }
9939 {
9940     Invalid~name.\\
9941     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9942     \SubMatrix\ of~your~\@@_full_name_env:.\\
9943     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9944     This~key~will~be~ignored.
9945 }
9946 \@@_msg_new:nn { Hbrace~not~allowed }
9947 {
9948     Command~not~allowed.\\
9949     You~can't~use~the~command~\token_to_str:N \Hbrace\

```

```

9950     because-you-have-not-loaded-TikZ-
9951     and-the-TikZ-library~'decorations.pathreplacing'.\\
9952     Use:~\token_to_str:N \usepackage{tikz}~
9953     \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9954     That-command-will-be-ignored.
9955 }
9956 \@@_msg_new:nn { Vbrace-not-allowed }
9957 {
9958     Command-not-allowed.\\
9959     You-can't-use-the-command~\token_to_str:N \Vbrace\
9960     because-you-have-not-loaded-TikZ-
9961     and-the-TikZ-library~'decorations.pathreplacing'.\\
9962     Use:~\token_to_str:N \usepackage{tikz}~
9963     \token_to_str:N \usetikzlibrary \{ decorations.pathreplacing \} \\
9964     That-command-will-be-ignored.
9965 }
9966 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9967 {
9968     Wrong-line.\\
9969     You-try-to-draw-a-#1-line-of-number~'#2'~in-a~
9970     \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
9971     number-is-not-valid.~It-will-be-ignored.
9972 }
9973 \@@_msg_new:nn { Impossible-delimiter }
9974 {
9975     Impossible-delimiter.\\
9976     It's-impossible-to-draw-the-#1-delimiter-of-your~
9977     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9978     in-that-column.
9979     \bool_if:NT \l_@@_submatrix_slim_bool
9980     { ~Maybe-you-should-try-without-the-key-'slim'. } \\
9981     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9982 }
9983 \@@_msg_new:nnn { width-without-X-columns }
9984 {
9985     You-have-used-the-key~'width'~but-you-have-put-no-'X'~column-in~
9986     the-preamble~(' \g_@@_user_preamble_tl' )~of~your~\@@_full_name_env:.\
9987     That-key-will-be-ignored.
9988 }
9989 {
9990     This-message-is-the-message-'width-without-X-columns'~
9991     of~the-module-'nicematrix'.~
9992     The-experimented-users-can-disable-that-message-with~
9993     \token_to_str:N \msg_redirect_name:nnn.\\
9994 }
9995
9996 \@@_msg_new:nn { key-multiplicity-with-dotted }
9997 {
9998     Incompatible-keys. \\
9999     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
10000     in-a-'custom-line'.~They-are-incompatible. \\
10001     The-key~'multiplicity'~will-be-discarded.
10002 }
10003 \@@_msg_new:nn { empty-environment }
10004 {
10005     Empty-environment.\\
10006     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
10007 }
10008 \@@_msg_new:nn { No-letter-and-no-command }
10009 {
10010     Erroneous-use.\\

```



```

10011 Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
10012 key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
10013 ~'ccommand'~(to~draw~horizontal~rules).\
10014 However,~you~can~go~on.
10015 }
10016 \@@_msg_new:nn { Forbidden~letter }
10017 {
10018   Forbidden~letter.\
10019   You~can't~use~the~letter~'#1'~for~a~customized~line.~
10020   It~will~be~ignored.\
10021   The~forbidden~letters~are:~\c_@@_forbidden_letters_str
10022 }
10023 \@@_msg_new:nn { Several~letters }
10024 {
10025   Wrong~name.\
10026   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
10027   have~used~'\l_@@_letter_str').\
10028   It~will~be~ignored.
10029 }
10030 \@@_msg_new:nn { Delimiter~with~small }
10031 {
10032   Delimiter~forbidden.\
10033   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
10034   because~the~key~'small'~is~in~force.\
10035   This~error~is~fatal.
10036 }
10037 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
10038 {
10039   Unknown~cell.\
10040   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
10041   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
10042   can't~be~executed~because~a~cell~doesn't~exist.\
10043   This~command~\token_to_str:N \line\ will~be~ignored.
10044 }
10045 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
10046 {
10047   Duplicate~name.\
10048   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
10049   in~this~\@@_full_name_env:.\
10050   This~key~will~be~ignored.\
10051   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10052   { For~a~list~of~the~names~already~used,~type~H<return>. }
10053 }
10054 {
10055   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
10056   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
10057 }
10058 \@@_msg_new:nn { r~or~l~with~preamble }
10059 {
10060   Erroneous~use.\
10061   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
10062   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
10063   your~\@@_full_name_env:.\
10064   This~key~will~be~ignored.
10065 }
10066 \@@_msg_new:nn { Hdotsfor~in~col~0 }
10067 {
10068   Erroneous~use.\
10069   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
10070   the~array.~This~error~is~fatal.
10071 }

```

```

10072 \@@_msg_new:nn { bad-corner }
10073 {
10074   Bad-corner.\
10075   #1-is-an-incorrect-specification-for-a-corner-(in-the-key~
10076   'corners').~The-available-values-are:~NW,~SW,~NE~and~SE.\
10077   This-specification-of-corner-will-be-ignored.
10078 }
10079 \@@_msg_new:nn { bad-border }
10080 {
10081   Bad-border.\
10082   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
10083   (in-the-key~'borders'~of-the-command~\token_to_str:N \Block).~
10084   The-available-values-are:~left,~right,~top~and~bottom~(and-you-can~
10085   also-use-the-key~'tikz'
10086   \IfPackageLoadedF { tikz }
10087     {~if-you-load-the-LaTeX-package~'tikz'}).
10088   This-specification-of-border-will-be-ignored.
10089 }
10090 \@@_msg_new:nn { TikzEveryCell~without~tikz }
10091 {
10092   TikZ-not-loaded.\
10093   You-can't-use~\token_to_str:N \TikzEveryCell\
10094   because-you~have~not~loaded-tikz.~
10095   This-command-will-be-ignored.
10096 }
10097 \@@_msg_new:nn { tikz-key~without~tikz }
10098 {
10099   TikZ-not-loaded.\
10100   You-can't-use-the-key~'tikz'~for-the-command~'\token_to_str:N
10101   \Block'~because-you-have~not~loaded-tikz.~
10102   This-key-will-be-ignored.
10103 }
10104 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
10105 {
10106   Erroneous-use.\
10107   In-the~\@@_full_name_env:,~you-must-use-the-key~
10108   'last-col'~without-value.\
10109   However,~you-can-go-on-for-this-time~
10110   (the-value~'\l_keys_value_tl'~will-be-ignored).
10111 }
10112 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
10113 {
10114   Erroneous-use.\
10115   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
10116   'last-col'~without-value.\
10117   However,~you-can-go-on-for-this-time~
10118   (the-value~'\l_keys_value_tl'~will-be-ignored).
10119 }
10120 \@@_msg_new:nn { Block-too-large-1 }
10121 {
10122   Block-too-large.\
10123   You-try-to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
10124   too-small~for~that~block. \
10125   This-block~and~maybe~others~will-be-ignored.
10126 }
10127 \@@_msg_new:nn { Block-too-large-2 }
10128 {
10129   Block-too-large.\
10130   The-preamble-of~your~\@@_full_name_env:\ announces~\int_use:N
10131   \g_@@_static_num_of_col_int\
10132   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~

```

```

10133     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands-
10134     (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
10135     This-block-and-maybe-others-will-be-ignored.
10136   }
10137   \@@_msg_new:nn { unknown-column-type }
10138   {
10139     Bad-column-type.\
10140     The-column-type-#1'~in-your~\@@_full_name_env:\
10141     is-unknown. \
10142     This-error-is-fatal.
10143   }
10144   \@@_msg_new:nn { unknown-column-type-S }
10145   {
10146     Bad-column-type.\
10147     The-column-type-'S'~in-your~\@@_full_name_env:\ is-unknown. \
10148     If-you-want-to-use-the-column-type-'S'~of-siunitx,~you-should-
10149     load-that-package. \
10150     This-error-is-fatal.
10151   }
10152   \@@_msg_new:nn { tabularnote-forbidden }
10153   {
10154     Forbidden-command.\
10155     You-can't-use-the-command~\token_to_str:N\tabularnote\
10156     ~here.~This-command-is-available-only-in-
10157     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in-
10158     the-argument-of-a-command~\token_to_str:N \caption\ included-
10159     in-an-environment~{table}. \
10160     This-command-will-be-ignored.
10161   }
10162   \@@_msg_new:nn { borders-forbidden }
10163   {
10164     Forbidden-key.\
10165     You-can't-use-the-key~'borders'~of~the-command~\token_to_str:N \Block\
10166     because-the-option~'rounded-corners'~
10167     is-in-force-with-a-non-zero-value.\
10168     This-key-will-be-ignored.
10169   }
10170   \@@_msg_new:nn { bottomrule-without-booktabs }
10171   {
10172     booktabs-not-loaded.\
10173     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
10174     loaded~'booktabs'.\
10175     This-key-will-be-ignored.
10176   }
10177   \@@_msg_new:nn { enumitem-not-loaded }
10178   {
10179     enumitem-not-loaded.\
10180     You-can't-use-the-command~\token_to_str:N\tabularnote\
10181     ~because-you-haven't~loaded~'enumitem'.\
10182     All-the-commands~\token_to_str:N\tabularnote\ will-be-
10183     ignored-in-the-document.
10184   }
10185   \@@_msg_new:nn { tikz-without-tikz }
10186   {
10187     Tikz-not-loaded.\
10188     You-can't-use-the-key~'tikz'~here~because~Tikz-is-not-
10189     loaded.~If-you-go-on,~that-key-will-be-ignored.
10190   }
10191   \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
10192   {

```

```

10193 Tikz~not~loaded.\\
10194 You~have~used~the~key~'tikz'~in~the~definition~of~a~
10195 customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
10196 You~can~go~on~but~you~will~have~another~error~if~you~actually~
10197 use~that~custom~line.
10198 }
10199 \@@_msg_new:nn { tikz~in~borders~without~tikz }
10200 {
10201 Tikz~not~loaded.\\
10202 You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
10203 command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
10204 That~key~will~be~ignored.
10205 }
10206 \@@_msg_new:nn { color~in~custom~line~with~tikz }
10207 {
10208 Erroneous~use.\\
10209 In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10210 which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10211 The~key~'color'~will~be~discarded.
10212 }
10213 \@@_msg_new:nn { Wrong~last~row }
10214 {
10215 Wrong~number.\\
10216 You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
10217 \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
10218 If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
10219 last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10220 without~value~(more~compilations~might~be~necessary).
10221 }
10222 \@@_msg_new:nn { Yet~in~env }
10223 {
10224 Nested~environments.\\
10225 Environments~of~nicematrix~can't~be~nested.\\
10226 This~error~is~fatal.
10227 }
10228 \@@_msg_new:nn { Outside~math~mode }
10229 {
10230 Outside~math~mode.\\
10231 The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
10232 (and~not~in~\token_to_str:N \vcenter).\\
10233 This~error~is~fatal.
10234 }
10235 \@@_msg_new:nn { One~letter~allowed }
10236 {
10237 Bad~name.\\
10238 The~value~of~key~'\l_keys_key_str'~must~be~of~length~1~and~
10239 you~have~used~'\l_keys_value_tl'.\\
10240 It~will~be~ignored.
10241 }
10242 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10243 {
10244 Environment~{TabularNote}~forbidden.\\
10245 You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
10246 but~*before*~the~\token_to_str:N \CodeAfter.\\
10247 This~environment~{TabularNote}~will~be~ignored.
10248 }
10249 \@@_msg_new:nn { varwidth~not~loaded }
10250 {
10251 varwidth~not~loaded.\\
10252 You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~

```

```

10253     loaded.\\
10254     Your~column~will~behave~like~'p'.
10255 }
10256 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
10257 {
10258     Unknown~key.\\
10259     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
10260     \c_@@_available_keys_str
10261 }
10262 {
10263     The~available~keys~are~(in~alphabetic~order):~
10264     color,~
10265     dotted,~
10266     multiplicity,~
10267     sep-color,~
10268     tikz,~and~total-width.
10269 }
10270
10271 \@@_msg_new:nnn { Unknown~key~for~Block }
10272 {
10273     Unknown~key.\\
10274     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10275     \Block.\\ It~will~be~ignored. \\
10276     \c_@@_available_keys_str
10277 }
10278 {
10279     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10280     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10281     opacity,~rounded-corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10282     and~vlines.
10283 }
10284 \@@_msg_new:nnn { Unknown~key~for~Brace }
10285 {
10286     Unknown~key.\\
10287     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10288     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10289     It~will~be~ignored. \\
10290     \c_@@_available_keys_str
10291 }
10292 {
10293     The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10294     right-shorten,~shorten~(which~fixes~both~left~shorten~and~
10295     right~shorten)~and~yshift.
10296 }
10297 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10298 {
10299     Unknown~key.\\
10300     The~key~'\l_keys_key_str'~is~unknown.\\
10301     It~will~be~ignored. \\
10302     \c_@@_available_keys_str
10303 }
10304 {
10305     The~available~keys~are~(in~alphabetic~order):~
10306     delimiters/color,~
10307     rules~(with~the~subkeys~'color'~and~'width'),~
10308     sub-matrix~(several~subkeys)~
10309     and~xdots~(several~subkeys).~
10310     The~latter~is~for~the~command~\token_to_str:N \line.
10311 }
10312 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10313 {
10314     Unknown~key.\\

```

```

10315 The-key~'\l_keys_key_str'~is~unknown.\\
10316 It~will~be~ignored. \\
10317 \c_@@_available_keys_str
10318 }
10319 {
10320 The-available-keys~are~(in~alphabetic~order):~
10321 create-cell-nodes,~
10322 delimiters/color~and~
10323 sub-matrix~(several~subkeys).
10324 }
10325 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10326 {
10327 Unknown~key.\\
10328 The-key~'\l_keys_key_str'~is~unknown.\\
10329 That~key~will~be~ignored. \\
10330 \c_@@_available_keys_str
10331 }
10332 {
10333 The-available-keys~are~(in~alphabetic~order):~
10334 'delimiters/color',~
10335 'extra-height',~
10336 'hlines',~
10337 'hvlines',~
10338 'left-xshift',~
10339 'name',~
10340 'right-xshift',~
10341 'rules'~(with~the~subkeys~'color'~and~'width'),~
10342 'slim',~
10343 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10344 and~'right-xshift').\\
10345 }
10346 \@@_msg_new:nnn { Unknown~key~for~notes }
10347 {
10348 Unknown~key.\\
10349 The-key~'\l_keys_key_str'~is~unknown.\\
10350 That~key~will~be~ignored. \\
10351 \c_@@_available_keys_str
10352 }
10353 {
10354 The-available-keys~are~(in~alphabetic~order):~
10355 bottomrule,~
10356 code-after,~
10357 code-before,~
10358 detect-duplicates,~
10359 enumitem-keys,~
10360 enumitem-keys-para,~
10361 para,~
10362 label-in-list,~
10363 label-in-tabular~and~
10364 style.
10365 }
10366 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10367 {
10368 Unknown~key.\\
10369 The-key~'\l_keys_key_str'~is~unknown~for~the~command~
10370 \token_to_str:N \RowStyle. \\
10371 That~key~will~be~ignored. \\
10372 \c_@@_available_keys_str
10373 }
10374 {
10375 The-available-keys~are~(in~alphabetic~order):~
10376 bold,~
10377 cell-space-top-limit,~

```

```

10378     cell-space-bottom-limit,~
10379     cell-space-limits,~
10380     color,~
10381     fill~(alias:~rowcolor),~
10382     nb-rows,
10383     opacity~and~
10384     rounded-corners.
10385 }
10386 \@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10387 {
10388     Unknown~key.\\
10389     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10390     \token_to_str:N \NiceMatrixOptions. \\
10391     That~key~will~be~ignored. \\
10392     \c_@_available_keys_str
10393 }
10394 {
10395     The~available~keys~are~(in~alphabetic~order):~
10396     &~in~blocks,~
10397     allow~duplicate~names,~
10398     ampersand~in~blocks,~
10399     caption~above,~
10400     cell-space-bottom-limit,~
10401     cell-space-limits,~
10402     cell-space-top-limit,~
10403     code~for~first~col,~
10404     code~for~first~row,~
10405     code~for~last~col,~
10406     code~for~last~row,~
10407     corners,~
10408     custom~key,~
10409     create~extra~nodes,~
10410     create~medium~nodes,~
10411     create~large~nodes,~
10412     custom~line,~
10413     delimiters~(several~subkeys),~
10414     end~of~row,~
10415     first~col,~
10416     first~row,~
10417     hlines,~
10418     hvlines,~
10419     hvlines~except~borders,~
10420     last~col,~
10421     last~row,~
10422     left~margin,~
10423     light~syntax,~
10424     light~syntax~expanded,~
10425     matrix/columns~type,~
10426     no~cell~nodes,~
10427     notes~(several~subkeys),~
10428     nullify~dots,~
10429     pgf~node~code,~
10430     renew~dots,~
10431     renew~matrix,~
10432     respect~arraystretch,~
10433     rounded~corners,~
10434     right~margin,~
10435     rules~(with~the~subkeys~'color'~and~'width'),~
10436     small,~
10437     sub~matrix~(several~subkeys),~
10438     vl~lines,~
10439     xdots~(several~subkeys).
10440 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10441 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10442 {
10443   Unknown~key.\\
10444   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10445   \{NiceArray\}. \\
10446   That~key~will~be~ignored. \\
10447   \c_@@_available_keys_str
10448 }
10449 {
10450   The~available~keys~are~(in~alphabetic~order):~
10451   &~in~blocks,~
10452   ampersand~in~blocks,~
10453   b,~
10454   baseline,~
10455   c,~
10456   cell~space~bottom~limit,~
10457   cell~space~limits,~
10458   cell~space~top~limit,~
10459   code~after,~
10460   code~for~first~col,~
10461   code~for~first~row,~
10462   code~for~last~col,~
10463   code~for~last~row,~
10464   columns~width,~
10465   corners,~
10466   create~extra~nodes,~
10467   create~medium~nodes,~
10468   create~large~nodes,~
10469   extra~left~margin,~
10470   extra~right~margin,~
10471   first~col,~
10472   first~row,~
10473   hlines,~
10474   hvlines,~
10475   hvlines~except~borders,~
10476   last~col,~
10477   last~row,~
10478   left~margin,~
10479   light~syntax,~
10480   light~syntax~expanded,~
10481   name,~
10482   no~cell~nodes,~
10483   nullify~dots,~
10484   pgf~node~code,~
10485   renew~dots,~
10486   respect~arraystretch,~
10487   right~margin,~
10488   rounded~corners,~
10489   rules~(with~the~subkeys~'color'~and~'width'),~
10490   small,~
10491   t,~
10492   vlines,~
10493   xdots/color,~
10494   xdots/shorten~start,~
10495   xdots/shorten~end,~
10496   xdots/shorten~and~
10497   xdots/line~style.
10498 }

```

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).

```

10499 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }

```



```

10500 {
10501   Unknown~key.\\
10502   The~key~'\l_keys_key_str'~is~unknown~for~the~
10503   \@_full_name_env:. \\
10504   That~key~will~be~ignored. \\
10505   \c_@_available_keys_str
10506 }
10507 {
10508   The~available~keys~are~(in~alphabetic~order):~
10509   &~in~blocks,~
10510   ampersand~in~blocks,~
10511   b,~
10512   baseline,~
10513   c,~
10514   cell~space~bottom~limit,~
10515   cell~space~limits,~
10516   cell~space~top~limit,~
10517   code~after,~
10518   code~for~first~col,~
10519   code~for~first~row,~
10520   code~for~last~col,~
10521   code~for~last~row,~
10522   columns~type,~
10523   columns~width,~
10524   corners,~
10525   create~extra~nodes,~
10526   create~medium~nodes,~
10527   create~large~nodes,~
10528   extra~left~margin,~
10529   extra~right~margin,~
10530   first~col,~
10531   first~row,~
10532   hlines,~
10533   hvlines,~
10534   hvlines~except~borders,~
10535   l,~
10536   last~col,~
10537   last~row,~
10538   left~margin,~
10539   light~syntax,~
10540   light~syntax~expanded,~
10541   name,~
10542   no~cell~nodes,~
10543   nullify~dots,~
10544   pgf~node~code,~
10545   r,~
10546   renew~dots,~
10547   respect~arraystretch,~
10548   right~margin,~
10549   rounded~corners,~
10550   rules~(with~the~subkeys~'color'~and~'width'),~
10551   small,~
10552   t,~
10553   vlines,~
10554   xdots/color,~
10555   xdots/shorten~start,~
10556   xdots/shorten~end,~
10557   xdots/shorten~and~
10558   xdots/line~style.
10559 }
10560 \@_msg_new:nnn { Unknown~key~for~NiceTabular }
10561 {
10562   Unknown~key.\\

```

```

10563 The-key~'\l_keys_key_str'~is-unknown~for~the~environment~
10564 \{NiceTabular\}. \\
10565 That~key~will~be~ignored. \\
10566 \c_@@_available_keys_str
10567 }
10568 {
10569 The~available~keys~are~(in~alphabetic~order):~
10570 &~in~blocks,~
10571 ampersand~in~blocks,~
10572 b,~
10573 baseline,~
10574 c,~
10575 caption,~
10576 cell-space-bottom-limit,~
10577 cell-space-limits,~
10578 cell-space-top-limit,~
10579 code-after,~
10580 code-for-first-col,~
10581 code-for-first-row,~
10582 code-for-last-col,~
10583 code-for-last-row,~
10584 columns-width,~
10585 corners,~
10586 custom-line,~
10587 create-extra-nodes,~
10588 create-medium-nodes,~
10589 create-large-nodes,~
10590 extra-left-margin,~
10591 extra-right-margin,~
10592 first-col,~
10593 first-row,~
10594 hlines,~
10595 hvlines,~
10596 hvlines-except-borders,~
10597 label,~
10598 last-col,~
10599 last-row,~
10600 left-margin,~
10601 light-syntax,~
10602 light-syntax-expanded,~
10603 name,~
10604 no-cell-nodes,~
10605 notes~(several~subkeys),~
10606 nullify-dots,~
10607 pgf-node-code,~
10608 renew-dots,~
10609 respect-arraystretch,~
10610 right-margin,~
10611 rounded-corners,~
10612 rules~(with~the~subkeys~'color'~and~'width'),~
10613 short-caption,~
10614 t,~
10615 tabularnote,~
10616 vlines,~
10617 xdots/color,~
10618 xdots/shorten-start,~
10619 xdots/shorten-end,~
10620 xdots/shorten-and~
10621 xdots/line-style.
10622 }
10623 \@@_msg_new:nnn { Duplicate~name }
10624 {
10625 Duplicate~name.\\

```

```

10626 The-name-\l_keys_value_tl'-is-already-used-and-you-shouldn't-use-
10627 the-same-environment-name-twice.-You-can-go-on,-but,-
10628 maybe,-you-will-have-incorrect-results-especially-
10629 if-you-use-'columns-width=auto'.-If-you-don't-want-to-see-this-
10630 message-again,-use-the-key-'allow-duplicate-names'-in-
10631 '\token_to_str:N \NiceMatrixOptions'.\\
10632 \bool_if:NF \g_@@_messages_for_Overleaf_bool
10633 { For-a-list-of-the-names-already-used,-type-H<return>. }
10634 }
10635 {
10636 The-names-already-defined-in-this-document-are:-
10637 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10638 }
10639 \@@_msg_new:nn { Option~auto~for~columns~width }
10640 {
10641 Erroneous~use.\\
10642 You-can't-give-the-value-'auto'-to-the-key-'columns-width'-here.-~
10643 That-key-will-be-ignored.
10644 }
10645 \@@_msg_new:nn { NiceTabularX~without~X }
10646 {
10647 NiceTabularX~without~X.\\
10648 You-should-not-use-{NiceTabularX}~without~X~columns.\\
10649 However,-you-can-go-on.
10650 }
10651 \@@_msg_new:nn { Preamble~forgotten }
10652 {
10653 Preamble~forgotten.\\
10654 You-have-probably-forgotten-the-preamble-of-your~
10655 \@@_full_name_env:. \\
10656 This-error-is-fatal.
10657 }
10658 \@@_msg_new:nn { Invalid~col~number }
10659 {
10660 Invalid~column~number.\\
10661 A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10662 specifies~a~column~which~is~outside~the~array.-It~will~be~ignored.
10663 }
10664 \@@_msg_new:nn { Invalid~row~number }
10665 {
10666 Invalid~row~number.\\
10667 A~color~instruction~in~the~\token_to_str:N \CodeBefore\
10668 specifies~a~row~which~is~outside~the~array.-It~will~be~ignored.
10669 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	19
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	59
12	The redefinition of <code>\multicolumn</code>	74
13	The environment <code>{NiceMatrix}</code> and its variants	92
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
15	After the construction of the array	94
16	We draw the dotted lines	101
17	The actual instructions for drawing the dotted lines with Tikz	115
18	User commands available in the new environments	120
19	The command <code>\line</code> accessible in code-after	127
20	The command <code>\RowStyle</code>	128
21	Colors of cells, rows and columns	131
22	The vertical and horizontal rules	144
23	The empty corners	160
24	The environment <code>{NiceMatrixBlock}</code>	162
25	The extra nodes	164
26	The blocks	168
27	How to draw the dotted lines transparently	192
28	Automatic arrays	193
29	The redefinition of the command <code>\dotfill</code>	194
30	The command <code>\diagbox</code>	194

31	The keyword <code>\CodeAfter</code>	196
32	The delimiters in the preamble	197
33	The command <code>\SubMatrix</code>	198
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	207
35	The commands <code>HBrace</code> et <code>VBrace</code>	210
36	The command <code>TikzEveryCell</code>	212
37	The command <code>\ShowCellNames</code>	214
38	We process the options at package loading	216
39	About the package underscore	217
40	Error messages of the package	218