

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 24, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 6.29 of `nicematrix`, at the date of 2024/10/24.

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

```

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

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

```

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

```

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

```

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

```

35 \cs_new_protected:Npn \@@_error_or_warning:n
36 { \bool_if:NTF \g_@@_messages_for_Overleaf_bool \@@_warning:n \@@_error:n }

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

will be transformed in : $\F\{x=a,y=b,z=c,t=d\}\{arg\}$

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

89 \tl_const:Nn \c_@@_b_tl { b }
90 \tl_const:Nn \c_@@_c_tl { c }
91 \tl_const:Nn \c_@@_l_tl { l }
92 \tl_const:Nn \c_@@_r_tl { r }
93 \tl_const:Nn \c_@@_all_tl { all }
94 \tl_const:Nn \c_@@_dot_tl { . }
95 \tl_const:Nn \c_@@_default_tl { default }
96 \tl_const:Nn \c_@@_star_tl { * }
97 \str_const:Nn \c_@@_star_str { * }
98 \str_const:Nn \c_@@_r_str { r }
99 \str_const:Nn \c_@@_c_str { c }
100 \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).

```

101 \tl_new:N \l_@@_argspec_tl
102 \cs_generate_variant:Nn \seq_set_split:Nnn { N o }
103 \cs_generate_variant:Nn \str_lowercase:n { o }
104 \cs_generate_variant:Nn \str_set:Nn { N o }
105 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
106 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
107 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
108 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
109 \cs_generate_variant:Nn \dim_min:nn { v }
110 \cs_generate_variant:Nn \dim_max:nn { v }

111 \hook_gput_code:nnn { begindocument } { . }
112 {
113   \IfPackageLoadedTF { tikz }
114   {

```

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.

```

115   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
116   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
117 }
118 {
119   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
120   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
121 }
122 }

```

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`).

```

123 \IfClassLoadedTF { revtex4-1 }
124 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
125 {
126   \IfClassLoadedTF { revtex4-2 }
127   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
128   {

```

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

```

129     \cs_if_exist:NT \rvtx@ifformat@geq
130     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
131     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
132   }
133 }

```

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.

```

134 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
135 {
136   \iow_now:Nn \@mainaux
137   {
138     \ExplSyntaxOn
139     \cs_if_free:NT \pgfsyspdfmark
140     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
141     \ExplSyntaxOff
142   }
143   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
144 }

```

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.

```

145 \ProvideDocumentCommand \iddots { }
146 {
147   \mathinner
148   {
149     \tex_mkern:D 1 mu
150     \box_move_up:nn { 1 pt } { \hbox { . } }
151     \tex_mkern:D 2 mu
152     \box_move_up:nn { 4 pt } { \hbox { . } }
153     \tex_mkern:D 2 mu
154     \box_move_up:nn { 7 pt }
155     { \vbox:n { \kern 7 pt \hbox { . } } }
156     \tex_mkern:D 1 mu
157   }
158 }

```

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.

```

159 \hook_gput_code:nnn { begindocument } { . }
160 {
161   \IfPackageLoadedT { booktabs }
162   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
163 }
164 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
165 {
166   \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`).

```

167   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
168   {

```

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

```

169     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
170     { \@@_old_pgfulil@check@rerun { ##1 } { ##2 } }
171   }
172 }

```

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

```

173 \hook_gput_code:nnn { begindocument } { . }
174 {
175   \IfPackageLoadedF { colortbl }
176   {

```

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.

```

177     \cs_set_protected:Npn \CT@arc@ { }
178     \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
179     \cs_set_nopar:Npn \CT@arc #1 #2
180     {
181       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
182       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
183     }

```

Idem for `\CT@drs@`.

```

184     \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
185     \cs_set_nopar:Npn \CT@drs #1 #2
186     {
187       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
188       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
189     }
190     \cs_set_nopar:Npn \hline
191     {
192       \noalign { \ifnum 0 = ` } \fi
193       \cs_set_eq:NN \hskip \vskip
194       \cs_set_eq:NN \vrule \hrule
195       \cs_set_eq:NN \@width \@height
196       { \CT@arc@ \vline }
197       \futurelet \reserved@a
198       \@xhline
199     }
200   }
201 }

```

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.

```

202 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
203 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
204 {
205   \int_if_zero:nT \l_@@_first_col_int { \omit & }
206   \int_compare:nNnT { #1 } > \c_one_int
207   { \multispan { \int_eval:n { #1 - 1 } } & }
208   \multispan { \int_eval:n { #2 - #1 + 1 } }
209   {
210     \CT@arc@
211     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

212     \skip_horizontal:N \c_zero_dim
213   }

```

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`.

```

214   \everycr { }
215   \cr
216   \noalign { \skip_vertical:N -\arrayrulewidth }
217 }

```

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.

```

218 \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`.

```

219 { \@@_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*.

```

220 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
221 \cs_generate_variant:Nn \@@_cline_i:nn { e }
222 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
223 {
224   \tl_if_empty:nTF { #3 }
225     { \@@_cline_iii:w #1|#2-#2 \q_stop }
226     { \@@_cline_ii:w #1|#2-#3 \q_stop }
227 }
228 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
229 { \@@_cline_iii:w #1|#2-#3 \q_stop }
230 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
231 {

```

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).

```

232   \int_compare:nNnT { #1 } < { #2 }
233     { \multispan { \int_eval:n { #2 - #1 } } & }
234   \multispan { \int_eval:n { #3 - #2 + 1 } }
235   {
236     \CT@arc@
237     \leaders \hrule \@height \arrayrulewidth \hfill
238     \skip_horizontal:N \c_zero_dim
239   }

```

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

```

240   \peek_meaning_remove_ignore_spaces:NTF \cline
241     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
242     { \everycr { } \cr }
243 }

```

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

```

244 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

245 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
246 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
247 {
248   \tl_if_blank:nF { #1 }
249   {
250     \tl_if_head_eq_meaning:nNTF { #1 } [
251       { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
252       { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
253     ]
254 }

```

```

255 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
256 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
257 {
258   \tl_if_head_eq_meaning:nNTF { #1 } [
259     { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
260     { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
261   ]

```

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

```

262 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
263 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
264 {
265   \tl_if_head_eq_meaning:nNTF { #2 } [
266     { #1 #2 }
267     { #1 { #2 } }
268   ]

```

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

```

269 \cs_generate_variant:Nn \@@_color:n { o }
270 \cs_new_protected:Npn \@@_color:n #1
271 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

```

```

272 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
273 {
274   \tl_set_rescan:Nno
275     #1
276     {
277       \char_set_catcode_other:N >
278       \char_set_catcode_other:N <
279     }
280   #1
281 }

```

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.

```

282 \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).

```

283 \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.

```

284 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
285 { \int_use:N \g_@@_env_int }

```

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

```

286 \cs_new_protected:Npn \@@_qpoint:n #1
287 { \pgfpointanchor { \@@_env: - #1 } { center } }

```


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

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

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

```
289 \bool_new:N \g_@@_delims_bool
290 \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.

```
291 \bool_new:N \l_@@_preamble_bool
292 \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.

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

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

```
294 \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.

```
295 \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).

```
296 \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.).

```
297 \dim_new:N \l_@@_col_width_dim
298 \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.

```
299 \int_new:N \g_@@_row_total_int
300 \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).

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

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

```
302 \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.

```
303 \tl_new:N \l_@@_hpos_cell_tl
304 \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`.

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

Idem for the mono-row blocks.

```
306 \dim_new:N \g_@@_blocks_ht_dim
307 \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}`).

```
308 \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`.

```
309 \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.

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

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

```
311 \bool_new:N \l_@@_notes_detect_duplicates_bool
312 \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.

```
313 \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).

```
314 \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).

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

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

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

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

```
317 \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.

```
318 \bool_new:N \l_@@_X_bool
319 \bool_new:N \g_@@_caption_finished_bool
```

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

```
320 \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.

```
321 \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.

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

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

```
324 \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}`).

```
325 \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`).

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

For `\multicolumn`.

```
327 \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`.

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

```
329 \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 `:`.

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

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

```
332 \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`).

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

```
334 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
335 {
```

```
336   \if_mode_math: \else:
```

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

```
338   \fi:
```

```
339 }
```

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.

```
340 \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”.

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

```
342 \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`).

```
343 \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*.

```

344 \tl_new:N \g_@@_com_or_env_str
345 \tl_gset:Nn \g_@@_com_or_env_str { environment }

346 \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)`.

```

347 \cs_new:Npn \@@_full_name_env:
348 {
349   \str_if_eq:eeTF \g_@@_com_or_env_str { command }
350   { command \space \c_backslash_str \g_@@_name_env_str }
351   { environment \space \{ \g_@@_name_env_str \} }
352 }

```

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

```

353 \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.

```

354 \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.

```

355 \tl_new:N \g_@@_pre_code_before_tl
356 \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.

```

357 \tl_new:N \g_@@_pre_code_after_tl
358 \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.

```

359 \bool_new:N \l_@@_in_code_after_bool

```

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

```

360 \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.

```

361 \int_new:N \l_@@_old_iRow_int
362 \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).

```

363 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```
364 \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.

```
365 \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.

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

```
367 \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.

```
368 \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).

```
369 \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).

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

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

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

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

```
372 \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).

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

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

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

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

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

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

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

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

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

```

380 \dim_new:N \g_@@_dp_row_zero_dim
381 \dim_new:N \g_@@_ht_row_zero_dim
382 \dim_new:N \g_@@_ht_row_one_dim
383 \dim_new:N \g_@@_dp_ante_last_row_dim
384 \dim_new:N \g_@@_ht_last_row_dim
385 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

386 \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”.

```

387 \dim_new:N \g_@@_width_last_col_dim
388 \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.

```

389 \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!).

```

390 \seq_new:N \g_@@_pos_of_blocks_seq

```

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

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

```

391 \seq_new:N \g_@@_pos_of_xdots_seq

```

The sequence `\g_@@_pos_of_xdots_seq` will be used when we will draw the rules required by the key `hvlines` (these rules won’t be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key `draw=red!15` when using the command `\Block`). In that case, the rules specified, for instance, by `hvlines` must not be drawn around the block. That’s why we keep the information of all that stroken blocks in the following sequence.

```

392 \seq_new:N \g_@@_pos_of_stroken_blocks_seq

```

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```

393 \clist_new:N \l_@@_corners_cells_clist

```

The list of the names of the potential `\SubMatrix` in the `\CodeAfter` of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given `\SubMatrix`).

```

394 \seq_new:N \g_@@_submatrix_names_seq

```

The following flag will be raised if the key `width` is used in an environment `{NiceTabular}` (not in a command `\NiceMatrixOptions`). You use it to raise an error when this key is used while no column `X` is used.

```

395 \bool_new:N \l_@@_width_used_bool

```

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

```
396 \seq_new:N \g_@@_multicolumn_cells_seq
397 \seq_new:N \g_@@_multicolumn_sizes_seq
```

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the `\SubMatrix`—the `\SubMatrix` in the code-before).

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

The following counters will be used when drawing the rules.

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

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `\CodeBefore`. Each sub-matrix is represented by an “object” of the form `{i}{j}{k}{l}` where i and j are the number of row and column of the upper-left cell and k and l the number of row and column of the lower-right cell.

```
407 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

```
408 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `opacity`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

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

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by `nicematrix` when the key `corners` is used).

The following dimension corresponds to the key `rounded-corners` available in an individual environment `{NiceTabular}`. When that key is used, a clipping is applied in the `\CodeBefore` of the environment in order to have rounded corners for the potential colored panels.

```
415 \dim_new:N \l_@@_tab_rounded_corners_dim
```

The following token list correspond to the key `color` of the command `\Block` and also the key `color` of the command `\RowStyle`.

```
416 \tl_new:N \l_@@_color_tl
```

In the key `tikz` of a command `\Block` or in the argument of a command `\TikzEveryCell`, the final user puts a list of tikz keys. But, you have added another key, named `offset` (which means that an offset will be used for the frame of the block or the cell). The following parameter corresponds to that key.

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

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

```
418 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

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

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

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

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

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

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

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

The following flag corresponds to the keys `vlines` and `hlines` of the command `\Block` (the key `hvlines` is the conjunction of both).

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

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

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

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

The following key is set when the keys `hvlines` and `hvlines-except-borders` are used. It’s used only to change slightly the clipping path set by the key `rounded-corners` (for a `{tabular}`).

```
436 \bool_new:N \l_@@_hvlines_bool
```

The following flag will be used by (for instance) `\@@_vline_ii:`. When `\l_@@_dotted_bool` is true, a dotted line (with our system) will be drawn.

```
437 \bool_new:N \l_@@_dotted_bool
```

The following flag will be set to true during the composition of a caption specified (by the key `caption`).

```
438 \bool_new:N \l_@@_in_caption_bool
```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

- **First row**

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
439 \int_new:N \l_@@_first_row_int
440 \int_set:Nn \l_@@_first_row_int 1
```


- **First column**

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```
441 \int_new:N \l_@@_first_col_int
442 \int_set:NN \l_@@_first_col_int \c_one_int
```

- **Last row**

The counter `\l_@@_last_row_int` is the number of the potential “last row”, as specified by the key `last-row`. A value of `-2` means that there is no “last row”. A value of `-1` means that there is a “last row” but we don’t know the number of that row (the key `last-row` has been used without value and the actual value has not still been read in the `aux` file).

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

```
446 \bool_new:N \l_@@_last_col_without_value_bool
```

- **Last column**

For the potential “last column”, we use an integer. A value of `-2` means that there is no last column. A value of `-1` means that we are in an environment without preamble (e.g. `{bNiceMatrix}`) and there is a last column but we don’t know its value because the user has used the option `last-col` without value. A value of 0 means that the option `last-col` has been used in an environment with preamble (like `{pNiceArray}`): in this case, the key was necessary without argument. The command `\NiceMatrixOptions` also sets `\l_@@_last_col_int` to 0.

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

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

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

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

```
449 \bool_new:N \g_@@_last_col_found_bool
```

This boolean is set to `false` at the end of `\@@_pre_array_ii:`.

In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

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

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.

Some utilities

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

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

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

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

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

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

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

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

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

The following internal parameters are for:

- `\Ldots` with both extremities open (and hence also `\Hdotsfor` in an exterior row);
- `\Vdots` with both extremities open (and hence also `\Vdotsfor` in an exterior column);
- when the special character “:” is used in order to put the label of a so-called “dotted line” *on the line*, a margin of `\c_@@_innersep_middle_dim` will be added around the label.

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

5 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It's also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that's mandatory since that caption must be wrapped with a line width equal to the width 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.

```
481 \newcounter { tabularnote }
```

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

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

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

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

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

```
487 \seq_new:N \l_@@_notes_labels_seq
488 \newcounter{nicematrix_draft}
489 \cs_new_protected:Npn \@@_notes_format:n #1
490 {
491   \setcounter { nicematrix_draft } { #1 }
492   \@@_notes_style:n { nicematrix_draft }
493 }
```

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

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

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

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

```
495 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
```

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

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

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

```
497 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

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

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

The type of list `tabularnotes` will be used to format the tabular notes at the end of the array in the general case and `tabularnotes*` will be used if the key `para` is in force.

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

One must remind that we have allowed a `\tabular` in the caption and that caption may also be found in the list of tables (`\listoftables`). We want the command `\tabularnote` be no-op during the composition of that list. That's why we program `\tabularnote` to be no-op excepted in a floating environment or in an environment of `nicematrix`.

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

```

531         { #1 } { #2 }
532     }
533 }
534 }
535 }
536 {
537     \NewDocumentCommand \tabularnote { o m }
538     {
539         \@@_error_or_warning:n { enumitem~not~loaded }
540         \@@_gredirect_none:n { enumitem~not~loaded }
541     }
542 }
543 }
544 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
545 { \tl_if_novalue:nT { #1 } { #3 } }

```

For the version in normal conditions, that is to say not in the `caption`. `#1` is the optional argument of `\tabularnote` (maybe equal to the special marker expressed by `\c_novalue_tl`) and `#2` is the mandatory argument of `\tabularnote`.

```

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

```

You have to see whether the argument of `\tabularnote` has yet been used as argument of another `\tabularnote` in the same tabular. In that case, there will be only one note (for both commands `\tabularnote`) at the end of the tabular. We search the argument of our command `\tabularnote` in `\g_@@_notes_seq`. The position in the sequence will be stored in `\l_tmpa_int` (0 if the text is not in the sequence yet).

```

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

```

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

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

If the user have used `\tabularnote` without the optional argument, the `label` will be the special marker expressed by `\c_novalue_tl`.

When we will go through the sequence `\g_@@_notes_seq`, we will count in `\l_tmpb_int` the notes without explicit label in order to have the “current” value of the counter `\c@tabularnote`.

```

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

```

```

575     \@@_notes_format:n
576     {
577         \int_eval:n
578         {
579             \int_if_zero:nTF \l_tmpa_int
580             \c@tabularnote
581             \l_tmpa_int
582         }
583     }
584 }
585 { #1 }
586 }
587 \peek_meaning:NF \tabularnote
588 {

```

If the following token is *not* a `\tabularnote`, we have finished the sequence of successive commands `\tabularnote` and we have to format the labels of these tabular notes (in the array). We compose those labels in a box `\l_tmpa_box` because we will do a special construction in order to have this box in an overlapping position if we are at the end of a cell when `\l_@@_hpos_cell_tl` is equal to `c` or `r`.

```

589     \hbox_set:Nn \l_tmpa_box
590     {

```

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

```

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

```

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

```

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

```

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

```

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

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

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

```

Now the version when the command is used in the key `caption`. The main difficulty is that the argument of the command `\caption` is composed several times. In order to know the number of commands `\tabularnote` in the caption, we will consider that there should not be the same tabular note twice in the caption (in the main tabular, it's possible). Once we have found a tabular note which has yet been encountered, we consider that you are in a new composition of the argument of `\caption`.

```

614 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2

```

```

615 {
616   \bool_if:NTF \g_@@_caption_finished_bool
617   {
618     \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
619     { \int_gzero:N \c@tabularnote }

```

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

```

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

```

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

```

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

```

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

```

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

```

Now, we will compose the label of the footnote (in the caption). Even if we are not in the first composition, we have to compose that label!

```

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

```

6 Command for creation of rectangle nodes

The following command should be used in a `{pgfpicture}`. It creates a rectangle (empty but with a name).

#1 is the name of the node which will be created; **#2** and **#3** are the coordinates of one of the corner of the rectangle; **#4** and **#5** are the coordinates of the opposite corner.

```

648 \cs_new_protected:Npn \@@_pgf_rect_node:nmmnn #1 #2 #3 #4 #5
649 {
650   \begin { pgfscope }
651   \pgfset
652   {
653     inner~sep = \c_zero_dim ,
654     minimum~size = \c_zero_dim
655   }

```

```

656 \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
657 \pgfnode
658 { rectangle }
659 { center }
660 {
661   \vbox_to_ht:nn
662     { \dim_abs:n { #5 - #3 } }
663     {
664       \vfill
665       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
666     }
667   }
668 { #1 }
669 { }
670 \end { pgfscope }
671 }

```

The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgf_rect_node:nnnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

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

```

7 The options

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

```

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

```

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

```

698 \bool_new:N \l_@@_caption_above_bool

```


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

```
699 \bool_new:N \l_@@_color_inside_bool
```

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

```
700 \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`).

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

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

```
703 \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.

```
704 \dim_new:N \l_@@_xdots_inter_dim
705 \hook_gput_code:nnn { begindocument } { . }
706 { \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).

```
707 \dim_new:N \l_@@_xdots_shorten_start_dim
708 \dim_new:N \l_@@_xdots_shorten_end_dim
709 \hook_gput_code:nnn { begindocument } { . }
710 {
711   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
712   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
713 }
```

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.

```
714 \dim_new:N \l_@@_xdots_radius_dim
715 \hook_gput_code:nnn { begindocument } { . }
716 { \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.

```
717 \tl_new:N \l_@@_xdots_line_style_tl
718 \tl_const:Nn \c_@@_standard_tl { standard }
719 \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`.

```
720 \bool_new:N \l_@@_light_syntax_bool
721 \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).

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

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

```
724 \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`).

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

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

```
726 \bool_new:N \l_@@_parallelize_diags_bool
727 \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`.

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

```
729 \dim_new:N \l_@@_notes_above_space_dim
730 \hook_gput_code:nnn { begindocument } { . }
731 { \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.).

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

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

```
733 \cs_new_protected:Npn \@@_reset_arraystretch:
734 { \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).

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

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

```
736 \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.

```
737 \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”.

```
738 \bool_new:N \l_@@_medium_nodes_bool
739 \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).

```
740 \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`).

```
741 \dim_new:N \l_@@_left_margin_dim
742 \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`.

```
743 \dim_new:N \l_@@_extra_left_margin_dim
744 \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.

```
745 \tl_new:N \l_@@_end_of_row_tl
746 \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 “:”.

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

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

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

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

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

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

```

773     { \str_if_eq_p:nn { #1 } { standard } }
774     { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
775     { \@@_error:n { bad-option-for-line-style } }
776   } ,
777   line-style .value_required:n = true ,
778   color .tl_set:N = \l_@@_xdots_color_tl ,
779   color .value_required:n = true ,
780   radius .code:n =
781     \hook_gput_code:nnn { begindocument } { . }
782     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
783   radius .value_required:n = true ,
784   inter .code:n =
785     \hook_gput_code:nnn { begindocument } { . }
786     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
787   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}`.

```

788   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
789   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
790   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).

```

791   draw-first .code:n = \prg_do_nothing: ,
792   unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
793 }

```

```

794 \keys_define:nn { nicematrix / rules }
795 {
796   color .tl_set:N = \l_@@_rules_color_tl ,
797   color .value_required:n = true ,
798   width .dim_set:N = \arrayrulewidth ,
799   width .value_required:n = true ,
800   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
801 }

```

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

```

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

```

```

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

```

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

```

861         { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
862     }
863     { \@@_error:n { One~letter~allowed } }
864 } ,
865 vlines-in-sub-matrix .value_required:n = true ,
866 hvlines .code:n =
867     {
868         \bool_set_true:N \l_@@_hvlines_bool
869         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
870         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
871     } ,
872 hvlines-except-borders .code:n =
873     {
874         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
875         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
876         \bool_set_true:N \l_@@_hvlines_bool
877         \bool_set_true:N \l_@@_except_borders_bool
878     } ,
879 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.

```

880     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

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

```

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

```

904 \keys_define:nn { nicematrix / environments }
905   {
906     corners .clist_set:N = \l_@@_corners_clist ,
907     corners .default:n = { NW , SW , NE , SE } ,
908     code-before .code:n =
909       {
910         \tl_if_empty:nF { #1 }
911         {
912           \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
913           \bool_set_true:N \l_@@_code_before_bool
914         }
915       } ,
916     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}`.

```

917   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
918   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
919   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
920   baseline .tl_set:N = \l_@@_baseline_tl ,
921   baseline .value_required:n = true ,
922   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)`.

```

923   \str_if_eq:eeTF { #1 } { auto }
924   { \bool_set_true:N \l_@@_auto_columns_width_bool }
925   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
926   columns-width .value_required:n = true ,
927   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.

```

928   \legacy_if:nF { measuring@ }
929   {
930     \str_set:Ne \l_tmpa_str { #1 }
931     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str

```

```

932     { \@@_error:nn { Duplicate~name } { #1 } }
933     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
934     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
935   } ,
936   name .value_required:n = true ,
937   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
938   code-after .value_required:n = true ,
939   color-inside .code:n =
940     \bool_set_true:N \l_@@_color_inside_bool
941     \bool_set_true:N \l_@@_code_before_bool ,
942   color-inside .value_forbidden:n = true ,
943   colortbl-like .meta:n = color-inside
944 }
945 \keys_define:nn { nicematrix / notes }
946 {
947   para .bool_set:N = \l_@@_notes_para_bool ,
948   para .default:n = true ,
949   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
950   code-before .value_required:n = true ,
951   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
952   code-after .value_required:n = true ,
953   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
954   bottomrule .default:n = true ,
955   style .cs_set:Np = \@@_notes_style:n #1 ,
956   style .value_required:n = true ,
957   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
958   label-in-tabular .value_required:n = true ,
959   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
960   label-in-list .value_required:n = true ,
961   enumitem-keys .code:n =
962     {
963       \hook_gput_code:nnn { begindocument } { . }
964       {
965         \IfPackageLoadedT { enumitem }
966         { \setlist* [ tabularnotes ] { #1 } }
967       }
968     } ,
969   enumitem-keys .value_required:n = true ,
970   enumitem-keys-para .code:n =
971     {
972       \hook_gput_code:nnn { begindocument } { . }
973       {
974         \IfPackageLoadedT { enumitem }
975         { \setlist* [ tabularnotes* ] { #1 } }
976       }
977     } ,
978   enumitem-keys-para .value_required:n = true ,
979   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
980   detect-duplicates .default:n = true ,
981   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
982 }
983 \keys_define:nn { nicematrix / delimiters }
984 {
985   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
986   max-width .default:n = true ,
987   color .tl_set:N = \l_@@_delimiters_color_tl ,
988   color .value_required:n = true ,
989 }

```

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

```

990 \keys_define:nn { nicematrix }

```

```

991 {
992   NiceMatrixOptions .inherit:n =
993     { nicematrix / Global } ,
994   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
995   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
996   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
997   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
998   SubMatrix / rules .inherit:n = nicematrix / rules ,
999   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1000  CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1001  CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1002  NiceMatrix .inherit:n =
1003    {
1004      nicematrix / Global ,
1005      nicematrix / environments ,
1006    } ,
1007  NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1008  NiceMatrix / rules .inherit:n = nicematrix / rules ,
1009  NiceTabular .inherit:n =
1010    {
1011      nicematrix / Global ,
1012      nicematrix / environments
1013    } ,
1014  NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1015  NiceTabular / rules .inherit:n = nicematrix / rules ,
1016  NiceTabular / notes .inherit:n = nicematrix / notes ,
1017  NiceArray .inherit:n =
1018    {
1019      nicematrix / Global ,
1020      nicematrix / environments ,
1021    } ,
1022  NiceArray / xdots .inherit:n = nicematrix / xdots ,
1023  NiceArray / rules .inherit:n = nicematrix / rules ,
1024  pNiceArray .inherit:n =
1025    {
1026      nicematrix / Global ,
1027      nicematrix / environments ,
1028    } ,
1029  pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1030  pNiceArray / rules .inherit:n = nicematrix / rules ,
1031 }

```

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

```

1032 \keys_define:nn { nicematrix / NiceMatrixOptions }
1033 {
1034   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1035   delimiters / color .value_required:n = true ,
1036   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1037   delimiters / max-width .default:n = true ,
1038   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1039   delimiters .value_required:n = true ,
1040   width .dim_set:N = \l_@@_width_dim ,
1041   width .value_required:n = true ,
1042   last-col .code:n =
1043     \tl_if_empty:nF { #1 }
1044     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1045     \int_zero:N \l_@@_last_col_int ,
1046   small .bool_set:N = \l_@@_small_bool ,
1047   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.


```

1048   renew-matrix .code:n = \l_@@_renew_matrix: ,
1049   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}`.

```

1050   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.

```

1051   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)`.

```

1052   \str_if_eq:eeTF { #1 } { auto }
1053   { \l_@@_error:n { Option~auto~for~columns~width } }
1054   { \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.

```

1055   allow-duplicate-names .code:n =
1056   \l_@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1057   allow-duplicate-names .value_forbidden:n = true ,
1058   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1059   notes .value_required:n = true ,
1060   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1061   sub-matrix .value_required:n = true ,
1062   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1063   matrix / columns-type .value_required:n = true ,
1064   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1065   caption-above .default:n = true ,
1066   unknown .code:n = \l_@@_error:n { Unknown~key~for~NiceMatrixOptions }
1067 }

```

`\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.

```

1068 \NewDocumentCommand \NiceMatrixOptions { m }
1069 { \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.

```

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

```

```

1089     small .value_forbidden:n = true ,
1090     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1091 }

```

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

```

1092 \keys_define:nn { nicematrix / NiceArray }
1093 {

```

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.

```

1094     small .bool_set:N = \l_@@_small_bool ,
1095     small .value_forbidden:n = true ,
1096     last-col .code:n = \tl_if_empty:nF { #1 }
1097         { \@@_error:n { last-col-non~empty~for~NiceArray } }
1098         \int_zero:N \l_@@_last_col_int ,
1099     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1100     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1101     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1102 }

```

```

1103 \keys_define:nn { nicematrix / pNiceArray }
1104 {
1105     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1106     last-col .code:n = \tl_if_empty:nF { #1 }
1107         { \@@_error:n { last-col-non~empty~for~NiceArray } }
1108         \int_zero:N \l_@@_last_col_int ,
1109     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1110     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1111     delimiters / color .value_required:n = true ,
1112     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1113     delimiters / max-width .default:n = true ,
1114     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1115     delimiters .value_required:n = true ,
1116     small .bool_set:N = \l_@@_small_bool ,
1117     small .value_forbidden:n = true ,
1118     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1119     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1120     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1121 }

```

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

```

1122 \keys_define:nn { nicematrix / NiceTabular }
1123 {

```

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.

```

1124     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1125         \bool_set_true:N \l_@@_width_used_bool ,
1126     width .value_required:n = true ,
1127     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1128     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1129     tabularnote .value_required:n = true ,
1130     caption .tl_set:N = \l_@@_caption_tl ,
1131     caption .value_required:n = true ,
1132     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1133     short-caption .value_required:n = true ,
1134     label .tl_set:N = \l_@@_label_tl ,
1135     label .value_required:n = true ,
1136     last-col .code:n = \tl_if_empty:nF { #1 }
1137         { \@@_error:n { last-col-non~empty~for~NiceArray } }

```

```

1138             \int_zero:N \l_@@_last_col_int ,
1139     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1140     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1142 }

```

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
```

```

1143 \keys_define:nn { nicematrix / CodeAfter }
1144 {
1145     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1146     delimiters / color .value_required:n = true ,
1147     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1148     rules .value_required:n = true ,
1149     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1150     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1151     sub-matrix .value_required:n = true ,
1152     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1153 }

```

8 Important code used by `{NiceArrayWithDelims}`

The pseudo-environment `\@@_cell_begin:w-\@@_cell_end:` will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a `\halign` (via an environment `{array}`).

```

1154 \cs_new_protected:Npn \@@_cell_begin:w
1155 {

```

`\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.

```
1156     \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).

```
1157     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1158     \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.

```

1159     \int_compare:nNnT \c@jCol = \c_one_int
1160     { \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:`.

```
1161     \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1162     \@@_tuning_not_tabular_begin:
1163     \@@_tuning_first_row:
1164     \@@_tuning_last_row:
1165     \g_@@_row_style_tl
1166 }

```

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.

```
1167 \cs_new_protected:Npn \@@_tuning_first_row:
1168 {
1169   \if_int_compare:w \c@iRow = \c_zero_int
1170     \if_int_compare:w \c@jCol > \c_zero_int
1171       \l_@@_code_for_first_row_tl
1172       \xglobal \colorlet { nicematrix-first-row } { . }
1173     \fi:
1174   \fi:
1175 }
```

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.

```
1176 \cs_new_protected:Npn \@@_tuning_last_row:
1177 {
1178   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1179     \l_@@_code_for_last_row_tl
1180     \xglobal \colorlet { nicematrix-last-row } { . }
1181   \fi:
1182 }
```

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

```
1183 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1184 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1185 {
1186   \c_math_toggle_token
```

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

```
1187   \@@_tuning_key_small:
1188 }
1189 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token
```

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

```
1190 \cs_new_protected:Npn \@@_begin_of_row:
```

```

1191 {
1192   \int_gincr:N \c@iRow
1193   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1194   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1195   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1196   \pgfpicture
1197   \pgfrememberpicturepositiononpagetrue
1198   \pgfcoordinate
1199     { \@@_env: - row - \int_use:N \c@iRow - base }
1200     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1201   \str_if_empty:NF \l_@@_name_str
1202     {
1203       \pgfnodealias
1204         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1205         { \@@_env: - row - \int_use:N \c@iRow - base }
1206     }
1207   \endpgfpicture
1208 }

```

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

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

```

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

```

```

1247     }
1248   }
1249   \bool_gset_false:N \g_@@_rotate_bool
1250   \bool_gset_false:N \g_@@_rotate_c_bool
1251 }
1252 \cs_new_protected:Npn \@@_adjust_size_box:
1253 {
1254   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1255   {
1256     \box_set_wd:Nn \l_@@_cell_box
1257     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1258     \dim_gzero:N \g_@@_blocks_wd_dim
1259   }
1260   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1261   {
1262     \box_set_dp:Nn \l_@@_cell_box
1263     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1264     \dim_gzero:N \g_@@_blocks_dp_dim
1265   }
1266   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1267   {
1268     \box_set_ht:Nn \l_@@_cell_box
1269     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1270     \dim_gzero:N \g_@@_blocks_ht_dim
1271   }
1272 }
1273 \cs_new_protected:Npn \@@_cell_end:
1274 {

```

The following command is nullified in the tabulars.

```

1275   \@@_tuning_not_tabular_end:
1276   \hbox_set_end:
1277   \@@_cell_end_i:
1278 }
1279 \cs_new_protected:Npn \@@_cell_end_i:
1280 {

```

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.

```

1281   \g_@@_cell_after_hook_tl
1282   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1283   \@@_adjust_size_box:
1284   \box_set_ht:Nn \l_@@_cell_box
1285   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1286   \box_set_dp:Nn \l_@@_cell_box
1287   { \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”).

```

1288   \@@_update_max_cell_width:

```

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

```

1289   \@@_update_for_first_and_last_row:

```

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

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

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

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

```

1290   \bool_if:NTF \g_@@_empty_cell_bool
1291     { \box_use_drop:N \l_@@_cell_box }
1292     {
1293       \bool_if:NTF \g_@@_not_empty_cell_bool
1294         \@@_node_for_cell:
1295         {
1296           \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1297             \@@_node_for_cell:
1298             { \box_use_drop:N \l_@@_cell_box }
1299         }
1300     }
1301   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1302     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1303   \bool_gset_false:N \g_@@_empty_cell_bool
1304   \bool_gset_false:N \g_@@_not_empty_cell_bool
1305 }

```

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

```

1306 \cs_new_protected:Npn \@@_update_max_cell_width:
1307 {
1308   \dim_gset:Nn \g_@@_max_cell_width_dim
1309     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1310 }

```

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`).

```

1311 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1312 {
1313   \@@_math_toggle:
1314   \hbox_set_end:
1315   \bool_if:NF \g_@@_rotate_bool
1316     {
1317       \hbox_set:Nn \l_@@_cell_box
1318         {
1319           \makebox [ \l_@@_col_width_dim ] [ s ]
1320             { \hbox_unpack_drop:N \l_@@_cell_box }
1321         }
1322     }
1323   \@@_cell_end_i:
1324 }

```

```

1325 \pgfset
1326 {
1327   nicematrix / cell-node /.style =
1328   {
1329     inner-sep = \c_zero_dim ,
1330     minimum-width = \c_zero_dim
1331   }
1332 }

```

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

```

1333 \cs_new_protected:Npn \@@_node_for_cell:
1334 {
1335   \pgfpicture
1336   \pgfsetbaseline \c_zero_dim
1337   \pgfrememberpicturepositiononpagetrue
1338   \pgfset { nicematrix / cell-node }
1339   \pgfnode
1340   { rectangle }
1341   { base }
1342   {

```

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

```

1343     \set@color
1344     \box_use_drop:N \l_@@_cell_box
1345   }
1346   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1347   { \l_@@_pgf_node_code_tl }
1348   \str_if_empty:NF \l_@@_name_str
1349   {
1350     \pgfnodealias
1351     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1352     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1353   }
1354   \endpgfpicture
1355 }

```

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.

```

1356 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1357 {
1358   \cs_new_protected:Npn \@@_patch_node_for_cell:
1359   {
1360     \hbox_set:Nn \l_@@_cell_box
1361     {
1362       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1363       \hbox_overlap_left:n
1364       {
1365         \pgfsys@markposition
1366         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way `latex`, `divps`, `ps2pdf` (or Adobe Distiller). However, it seems to work.

```

1367         #1
1368       }
1369     \box_use:N \l_@@_cell_box
1370     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1371     \hbox_overlap_left:n
1372     {
1373       \pgfsys@markposition
1374       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1375     }
1376   }
1377 }
1378 }
1379 }

```

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

```

1380 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1381 {

```



```

1382 \@@_patch_node_for_cell:n
1383 { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1384 }
1385 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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).

```

1386 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1387 {
1388   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1389   { g_@@_#2 _ lines _ tl }
1390   {
1391     \use:c { @@ _ draw _ #2 : nnn }
1392     { \int_use:N \c@iRow }
1393     { \int_use:N \c@jCol }
1394     { \exp_not:n { #3 } }
1395   }
1396 }

```

```

1397 \cs_generate_variant:Nn \@@_array:n { o }
1398 \cs_new_protected:Npn \@@_array:n
1399 {
1400 % \begin{macrocode}
1401 \dim_set:Nn \col@sep
1402 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1403 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1404 { \cs_set_nopar:Npn \@halignto { } }
1405 { \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`.

```

1406 \@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)`.

```

1407 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1408 }

```

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`.

```

1409 \bool_if:NTF \c_@@_tagging_array_bool
1410 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1411 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

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

1420 \cs_new_protected:Npn \@@_create_row_node_i:
1421 {

```

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

```

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

```

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

```

1447 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1448 \cs_new_protected:Npn \@@_everycr_i:
1449 {
1450   \bool_if:NT \c_@@_testphase_table_bool
1451   {
1452     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1453     \tbl_update_cell_data_for_next_row:
1454   }
1455   \int_gzero:N \c@jCol
1456   \bool_gset_false:N \g_@@_after_col_zero_bool
1457   \bool_if:NF \g_@@_row_of_col_done_bool
1458   {
1459     \@@_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).

```

1460     \clist_if_empty:NF \l_@@_hlines_clist
1461     {
1462       \str_if_eq:eeF \l_@@_hlines_clist { all }
1463       {
1464         \clist_if_in:NeT

```

```

1465         \l_@@_hlines_clist
1466         { \int_eval:n { \c@iRow + 1 } }
1467     }
1468     {

```

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.

```

1469         \int_compare:nNnT \c@iRow > { -1 }
1470         {
1471             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1472             { \hrule height \arrayrulewidth width \c_zero_dim }
1473         }
1474     }
1475 }
1476 }
1477 }

```

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

```

1478 \cs_set_protected:Npn \@@_renew_dots:
1479 {
1480     \cs_set_eq:NN \ldots \@@_Ldots
1481     \cs_set_eq:NN \cdots \@@_Cdots
1482     \cs_set_eq:NN \vdots \@@_Vdots
1483     \cs_set_eq:NN \ddots \@@_Ddots
1484     \cs_set_eq:NN \iddots \@@_Iddots
1485     \cs_set_eq:NN \dots \@@_Ldots
1486     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1487 }
1488 \cs_new_protected:Npn \@@_test_color_inside:
1489 {
1490     \bool_if:NF \l_@@_color_inside_bool
1491     {

```

We will issue an error only during the first run.

```

1492         \bool_if:NF \g_@@_aux_found_bool
1493         { \@@_error:n { without~color~inside } }
1494     }
1495 }

```

```

1496 \cs_new_protected:Npn \@@_redefine_everycr:
1497 { \everycr { \@@_everycr: } }
1498 \hook_gput_code:nnn { begindocument } { . }
1499 {
1500     \IfPackageLoadedT { colortbl }
1501     {
1502         \cs_set_protected:Npn \@@_redefine_everycr:
1503         {
1504             \CT@everycr
1505             {
1506                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1507                 \@@_everycr:
1508             }
1509         }
1510     }
1511 }

```

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.

```

1512 \hook_gput_code:nnn { begindocument } { . }
1513 {
1514   \IfPackageLoadedTF { booktabs }
1515   {
1516     \cs_new_protected:Npn \@@_patch_booktabs:
1517       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1518   }
1519   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1520 }

```

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`.

```

1521 \cs_new_protected:Npn \@@_some_initialization:
1522 {
1523   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1524   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1525   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1526   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1527   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1528   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1529 }

```

#1468 on GitHub (latex2e) exhibits a bug when a command with a last argument which is optional is used in `>{...}` of `array`.

In order to solve that problem, a modification is done to the command `\inser@column` in the version 2.6g (2024/10/12) of `array`. However, with that modification, our command `\@@_test_if_empty:` leads to a bug when the first column of an array is of type `p{...}` and when the first cell of that first column is empty.

As a workaround, we will use a slight different version of `\insert@column`.

```

1530 \bool_if:NT \c_@@_tagging_array_bool
1531 {
1532   \cs_new:Npn \@@_insert@column
1533   {
1534     \UseTaggingSocket{tbl/cell/begin}
1535     \the@toks \the \@tempcnta \ignorespaces
1536     \@sharp \textonly@unskip
1537     \the@toks \the \count@ \relax
1538     \UseTaggingSocket{tbl/cell/end}
1539   }
1540 }

```

In the version 2.6g of `array`, a command `\@protected@firstofone` is added:
`\@protected@firstofone { \the@toks \the \@tempcnta \ignorespaces }`

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

```

1541 \cs_new_protected:Npn \@@_pre_array_ii:
1542 {
1543   \bool_if:NT \c_@@_tagging_array_bool
1544   { \cs_set_eq:NN \insert@column \@@_insert@column }

```

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}`.

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

```

1545 \int_gzero:N \g_@@_total_X_weight_int
1546 \@@_expand_clist:N \l_@@_hlines_clist
1547 \@@_expand_clist:N \l_@@_vlines_clist
1548 \@@_patch_booktabs:
1549 \box_clear_new:N \l_@@_cell_box
1550 \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}`).

```

1551 \bool_if:NT \l_@@_small_bool
1552 {
1553 \cs_set_nopar:Npn \arraystretch { 0.47 }
1554 \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1555 \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1556 }

1557 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1558 {
1559 \tbl_put_right:Nn \@@_begin_of_row:
1560 {
1561 \pgfsys@markposition
1562 { \@@_env: - row - \int_use:N \c@iRow - base }
1563 }
1564 }

```

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

```

1565 \bool_if:NTF \c_@@_tagging_array_bool
1566 {
1567 \cs_set_nopar:Npn \ar@ialign
1568 {
1569 \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1570 \@@_redefine_everycr:
1571 \dim_zero:N \tabskip
1572 \@@_some_initialization:

```

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

```

1573 \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1574 \halign
1575 }
1576 }

```

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

```

1577 {
1578 \cs_set_nopar:Npn \ialign
1579 {
1580 \@@_redefine_everycr:
1581 \dim_zero:N \tabskip
1582 \@@_some_initialization:
1583 \cs_set_eq:NN \ialign \@@_old_ialign:
1584 \halign
1585 }
1586 }

```

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).

```

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

```

1623 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1624 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1625   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1626 \@@_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!).

```

1627 \tl_if_exist:NT \l_@@_note_in_caption_tl
1628   {
1629     \tl_if_empty:NF \l_@@_note_in_caption_tl
1630       {
1631         \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1632         \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1633       }
1634   }

```

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).

```
1635 \seq_gclear:N \g_@@_multicolumn_cells_seq
1636 \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).

```
1637 \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`).

```
1638 \int_gzero_new:N \g_@@_row_total_int
```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:w` executed at the beginning of each cell.

```
1639 \int_gzero_new:N \g_@@_col_total_int
1640 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1641 \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.

```
1642 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1643 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1644 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1645 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1646 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1647 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1648 \tl_gclear:N \g_nicematrix_code_before_tl
1649 \tl_gclear:N \g_@@_pre_code_before_tl
1650 }
```

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

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

```
1651 \cs_new_protected:Npn \@@_pre_array:
1652 {
1653 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1654 \int_gzero_new:N \c@iRow
1655 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1656 \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).

```
1657 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1658 {
1659 \bool_set_true:N \l_@@_last_row_without_value_bool
1660 \bool_if:NT \g_@@_aux_found_bool
1661 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1662 }
1663 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1664 {
1665 \bool_if:NT \g_@@_aux_found_bool
1666 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1667 }
```

If there is an exterior row, we patch a command used in `\@@_cell_begin:w` in order to keep track of some dimensions needed to the construction of that “last row”.

```

1668   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1669   {
1670     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1671     {
1672       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1673       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1674       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1675       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1676     }
1677   }

1678   \seq_gclear:N \g_@@_cols_vlism_seq
1679   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1680   \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.

```

1681   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1682   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1683   \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.

```

1684   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1685   \@@_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.

```

1686   \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]`).

```

1687   \dim_zero_new:N \l_@@_left_delim_dim
1688   \dim_zero_new:N \l_@@_right_delim_dim
1689   \bool_if:NTF \g_@@_delims_bool
1690   {

```

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

```

1691     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1692     \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1693     \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1694     \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1695   }
1696   {
1697     \dim_gset:Nn \l_@@_left_delim_dim
1698     { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1699     \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1700   }

```


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).

```

1701   \hbox_set:Nw \l_@@_the_array_box
1702   \bool_if:NT \c_@@_testphase_table_bool
1703     { \UseTaggingSocket { tbl / hmode / begin } }

1704   \skip_horizontal:N \l_@@_left_margin_dim
1705   \skip_horizontal:N \l_@@_extra_left_margin_dim
1706   \c_math_toggle_token
1707   \bool_if:NTF \l_@@_light_syntax_bool
1708     { \use:c { @@-light-syntax } }
1709     { \use:c { @@-normal-syntax } }
1710 }

```

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

```

1711 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1712 {
1713   \tl_set:Nn \l_tmpa_tl { #1 }
1714   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1715     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1716   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1717   \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...

```

1718   \@@_pre_array:
1719 }

```

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).

```

1720 \cs_new_protected:Npn \@@_pre_code_before:
1721 {

```

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).

```

1722   \int_set:Nn \c_iRow { \seq_item:Nn \g_@@_size_seq 2 }
1723   \int_set:Nn \c_jCol { \seq_item:Nn \g_@@_size_seq 5 }
1724   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1725   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

1726   \pgfsys@markposition { \@@_env: - position }
1727   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1728   \pgfpicture
1729   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1730 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1731 {
1732   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1733   \pgfcoordinate { \@@_env: - row - ##1 }
1734   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1735 }

```

Now, the recreation of the col nodes.

```

1736 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1737 {
1738   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1739   \pgfcoordinate { \@@_env: - col - ##1 }
1740   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1741 }

```

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

```

1742 \@@_create_diag_nodes:

```

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

```

1743 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1744 \endpgfpicture

```

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

```

1745 \@@_create_blocks_nodes:
1746 \IfPackageLoadedT { tikz }
1747 {
1748   \tikzset
1749   {
1750     every-picture / .style =
1751     { overlay , name~prefix = \@@_env: - }
1752   }
1753 }
1754 \cs_set_eq:NN \cellcolor \@@_cellcolor
1755 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1756 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1757 \cs_set_eq:NN \rowcolor \@@_rowcolor
1758 \cs_set_eq:NN \rowcolors \@@_rowcolors
1759 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1760 \cs_set_eq:NN \arraycolor \@@_arraycolor
1761 \cs_set_eq:NN \columncolor \@@_columncolor
1762 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1763 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1764 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1765 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1766 }

```

```

1767 \cs_new_protected:Npn \@@_exec_code_before:
1768 {

```

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...

```

1769 \clist_map_inline:Nn \l_@@_corners_cells_clist
1770 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1771 \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`.

```

1772 \@@_add_to_colors_seq:nn { { nocolor } } { }
1773 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1774 \group_begin:

```

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

```
1775 \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`).

```
1776 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1777 { \@@_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`.

```
1778 \exp_last_unbraced:No \@@_CodeBefore_keys:
1779 \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.

```
1780 \@@_actually_color:
1781 \l_@@_code_before_tl
1782 \q_stop
1783 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1784 \group_end:
1785 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1786 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1787 }
```

```
1788 \keys_define:nn { nicematrix / CodeBefore }
1789 {
1790   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1791   create-cell-nodes .default:n = true ,
1792   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1793   sub-matrix .value_required:n = true ,
1794   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1795   delimiters / color .value_required:n = true ,
1796   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1797 }

1798 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1799 {
1800   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1801   \@@_CodeBefore:w
1802 }
```

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.

```
1803 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1804 {
1805   \bool_if:NT \g_@@_aux_found_bool
1806   {
1807     \@@_pre_code_before:
1808     #1
1809   }
1810 }
```

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.

```

1811 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1812 {
1813   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1814   {
1815     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1816     \pgfcoordinate { \@@_env: - row - ##1 - base }
1817     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1818     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1819     {
1820       \cs_if_exist:cT
1821       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1822       {
1823         \pgfsys@getposition
1824         { \@@_env: - ##1 - #####1 - NW }
1825         \@@_node_position:
1826         \pgfsys@getposition
1827         { \@@_env: - ##1 - #####1 - SE }
1828         \@@_node_position_i:
1829         \@@_pgf_rect_node:nnn
1830         { \@@_env: - ##1 - #####1 }
1831         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1832         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1833       }
1834     }
1835   }
1836   \int_step_inline:nn \c@iRow
1837   {
1838     \pgfnodealias
1839     { \@@_env: - ##1 - last }
1840     { \@@_env: - ##1 - \int_use:N \c@jCol }
1841   }
1842   \int_step_inline:nn \c@jCol
1843   {
1844     \pgfnodealias
1845     { \@@_env: - last - ##1 }
1846     { \@@_env: - \int_use:N \c@iRow - ##1 }
1847   }
1848   \@@_create_extra_nodes:
1849 }

```

```

1850 \cs_new_protected:Npn \@@_create_blocks_nodes:
1851 {
1852   \pgfpicture
1853   \pgf@relevantforpicturesizefalse
1854   \pgfrememberpicturepositiononpagetrue
1855   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1856   { \@@_create_one_block_node:nnnnn ##1 }
1857   \endpgfpicture
1858 }

```

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

```

1859 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1860 {
1861   \tl_if_empty:nF { #5 }
1862   {
1863     \@@_qpoint:n { col - #2 }

```

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).

```

1864     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1865     \@@_qpoint:n { #1 }
1866     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1867     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1868     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1869     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1870     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1871     \@@_pgf_rect_node:nnnnn
1872     { \@@_env: - #5 }
1873     { \dim_use:N \l_tmpa_dim }
1874     { \dim_use:N \l_tmpb_dim }
1875     { \dim_use:N \l_@@_tmpc_dim }
1876     { \dim_use:N \l_@@_tmpd_dim }
1877   }
1878 }

1879 \cs_new_protected:Npn \@@_patch_for_revtex:
1880 {
1881   \cs_set_eq:NN \@@addamp \@@addamp@LaTeX
1882   \cs_set_eq:NN \@@insert@column \@@insert@column@array
1883   \cs_set_eq:NN \@@classx \@@classx@array
1884   \cs_set_eq:NN \@@xarraycr \@@xarraycr@array
1885   \cs_set_eq:NN \@@arraycr \@@arraycr@array
1886   \cs_set_eq:NN \@@xargarraycr \@@xargarraycr@array
1887   \cs_set_eq:NN \@@array \@@array@array
1888   \cs_set_eq:NN \@@array \@@array@array
1889   \cs_set_eq:NN \@@tabular \@@tabular@array
1890   \cs_set_eq:NN \@@mkpream \@@mkpream@array
1891   \cs_set_eq:NN \@@endarray \@@endarray@array
1892   \cs_set:Npn \@@tabarray { \@@ifnextchar [ { \@@array } { \@@array [ c ] } }
1893   \cs_set:Npn \@@endtabular { \@@endarray $\egroup} % $
1894 }

```

10 The environment `{NiceArrayWithDelims}`

```

1895 \NewDocumentEnvironment { NiceArrayWithDelims }
1896 { m m 0 { } m ! 0 { } t \CodeBefore }
1897 {
1898   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1899   \@@_provide_pgfsyspdfmark:
1900   \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

1901   \bgroup

1902   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1903   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1904   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1905   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1906   \int_gzero:N \g_@@_block_box_int
1907   \dim_zero:N \g_@@_width_last_col_dim
1908   \dim_zero:N \g_@@_width_first_col_dim
1909   \bool_gset_false:N \g_@@_row_of_col_done_bool
1910   \str_if_empty:NT \g_@@_name_env_str
1911     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1912   \bool_if:NTF \l_@@_tabular_bool

```

```

1913     \mode_leave_vertical:
1914     \@@_test_if_math_mode:
1915     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1916     \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.

```

1917     \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

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

```

1918     \cs_if_exist:NT \tikz@library@external@loaded
1919     {
1920         \tikzexternaldisable
1921         \cs_if_exist:NT \ifstandalone
1922         { \tikzset { external / optimize = false } }
1923     }

```

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

```

1924     \int_gincr:N \g_@@_env_int
1925     \bool_if:NF \l_@@_block_auto_columns_width_bool
1926     { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1927     \seq_gclear:N \g_@@_blocks_seq
1928     \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`.

```

1929     \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1930     \seq_gclear:N \g_@@_pos_of_xdots_seq
1931     \tl_gclear_new:N \g_@@_code_before_tl
1932     \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.

```

1933     \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1934     {
1935         \bool_gset_true:N \g_@@_aux_found_bool
1936         \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1937     }
1938     { \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.

```

1939     \tl_gclear:N \g_@@_aux_tl
1940     \tl_if_empty:NF \g_@@_code_before_tl
1941     {
1942         \bool_set_true:N \l_@@_code_before_bool
1943         \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1944     }
1945     \tl_if_empty:NF \g_@@_pre_code_before_tl
1946     { \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`.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1947 \bool_if:NTF \g_@@_delims_bool
1948   { \keys_set:nn { nicematrix / pNiceArray } }
1949   { \keys_set:nn { nicematrix / NiceArray } }
1950   { #3 , #5 }

1951 \@@_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:.

```

1952 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1953 }

```

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

```

1954 {
1955   \bool_if:NTF \l_@@_light_syntax_bool
1956     { \use:c { end @@-light-syntax } }
1957     { \use:c { end @@-normal-syntax } }
1958   \c_math_toggle_token
1959   \skip_horizontal:N \l_@@_right_margin_dim
1960   \skip_horizontal:N \l_@@_extra_right_margin_dim
1961
1962   % awful workaround
1963   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1964     {
1965       \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1966         {
1967           \skip_horizontal:N - \l_@@_columns_width_dim
1968           \bool_if:NTF \l_@@_tabular_bool
1969             { \skip_horizontal:n { - 2 \tabcolsep } }
1970             { \skip_horizontal:n { - 2 \arraycolsep } }
1971         }
1972     }
1973   \hbox_set_end:

```

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

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

```

1974 \bool_if:NT \l_@@_width_used_bool
1975   {
1976     \int_if_zero:nT \g_@@_total_X_weight_int
1977       { \@@_error_or_warning:n { width-without-X-columns } }
1978   }

```

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 .

```

1979 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1980   {
1981     \tl_gput_right:Ne \g_@@_aux_tl
1982     {
1983       \bool_set_true:N \l_@@_X_columns_aux_bool
1984       \dim_set:Nn \l_@@_X_columns_dim
1985         {
1986           \dim_compare:nNnTF
1987             {
1988               \dim_abs:n
1989                 { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1990             }
1991             <

```

```

1992         { 0.001 pt }
1993         { \dim_use:N \l_@@_X_columns_dim }
1994         {
1995           \dim_eval:n
1996           {
1997             ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1998             / \int_use:N \g_@@_total_X_weight_int
1999             + \l_@@_X_columns_dim
2000           }
2001         }
2002       }
2003     }
2004   }

```

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

```

2005   \int_compare:nNnT \l_@@_last_row_int > { -2 }
2006   {
2007     \bool_if:NF \l_@@_last_row_without_value_bool
2008     {
2009       \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2010       {
2011         \@@_error:n { Wrong-last-row }
2012         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2013       }
2014     }
2015   }

```

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

```

2016   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2017   \bool_if:NTF \g_@@_last_col_found_bool
2018   { \int_gdecr:N \c@jCol }
2019   {
2020     \int_compare:nNnT \l_@@_last_col_int > { -1 }
2021     { \@@_error:n { last-col-not-used } }
2022   }

```

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

```

2023   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2024   \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2025   \int_if_zero:nT \l_@@_first_col_int
2026   { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2027   \bool_if:nTF { ! \g_@@_delims_bool }
2028   {
2029     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2030     \@@_use_arraybox_with_notes_c:
2031     {
2032       \str_if_eq:eeTF \l_@@_baseline_tl { b }
2033       \@@_use_arraybox_with_notes_b:
2034       \@@_use_arraybox_with_notes:
2035     }
2036   }

```

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

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).

```

2037     {
2038         \int_if_zero:nTF \l_@@_first_row_int
2039         {
2040             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2041             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2042         }
2043         { \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

```

2044         \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2045         {
2046             \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2047             \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2048         }
2049         { \dim_zero:N \l_tmpb_dim }
2050     \hbox_set:Nn \l_tmpa_box
2051     {
2052         \c_math_toggle_token
2053         \@@_color:o \l_@@_delimiters_color_tl
2054         \exp_after:wN \left \g_@@_left_delim_tl
2055         \vcenter
2056         {

```

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.

```

2057             \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2058             \hbox
2059             {
2060                 \bool_if:NTF \l_@@_tabular_bool
2061                 { \skip_horizontal:N -\tabcolsep }
2062                 { \skip_horizontal:N -\arraycolsep }
2063                 \@@_use_arraybox_with_notes_c:
2064                 \bool_if:NTF \l_@@_tabular_bool
2065                 { \skip_horizontal:N -\tabcolsep }
2066                 { \skip_horizontal:N -\arraycolsep }
2067             }

```

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

```

2068             \skip_vertical:N -\l_tmpb_dim
2069             \skip_vertical:N \arrayrulewidth
2070         }
2071         \exp_after:wN \right \g_@@_right_delim_tl
2072         \c_math_toggle_token
2073     }

```

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.

```

2074         \bool_if:NTF \l_@@_delimiters_max_width_bool
2075         {
2076             \@@_put_box_in_flow_bis:nn
2077             \g_@@_left_delim_tl
2078             \g_@@_right_delim_tl
2079         }
2080         \@@_put_box_in_flow:
2081     }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 90).

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).

```

2082 \bool_if:NT \g_@@_last_col_found_bool
2083   { \skip_horizontal:N \g_@@_width_last_col_dim }
2084 \bool_if:NT \l_@@_preamble_bool
2085   {
2086     \int_compare:nNt \c@jCol < \g_@@_static_num_of_col_int
2087     { \@@_warning_gredirect_none:n { columns~not~used } }
2088   }
2089 \@@_after_array:

```

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

```

2090 \egroup

```

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

```

2091 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2092 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2093 \iow_now:Ne \@mainaux
2094   {
2095     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2096     { \exp_not:o \g_@@_aux_tl }
2097   }
2098 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2099 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2100 }

```

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

11 We construct 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.

```

2101 \cs_new_protected:Npn \@@_transform_preamble:
2102   {
2103     \@@_transform_preamble_i:
2104     \@@_transform_preamble_ii:
2105   }

2106 \cs_new_protected:Npn \@@_transform_preamble_i:
2107   {
2108     \int_gzero:N \c@jCol

```

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

```

2109 \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.

```

2110 \bool_gset_false:N \g_tmpb_bool

```

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

```

2111 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2112 \int_zero:N \l_tmpa_int
2113 \tl_gclear:N \g_@@_array_preamble_tl
2114 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2115   {
2116     \tl_gset:Nn \g_@@_array_preamble_tl

```

```

2117     { ! { \skip_horizontal:N \arrayrulewidth } }
2118   }
2119   {
2120     \clist_if_in:NnT \l_@@_vlines_clist 1
2121     {
2122       \tl_gset:Nn \g_@@_array_preamble_tl
2123         { ! { \skip_horizontal:N \arrayrulewidth } }
2124     }
2125   }

```

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

```

2126     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2127     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2128     \@@_replace_columncolor:
2129   }

2130 \hook_gput_code:nnn { begindocument } { . }
2131 {
2132   \IfPackageLoadedTF { colortbl }
2133   {

```

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).

```

2134     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2135     \cs_new_protected:Npn \@@_replace_columncolor:
2136     {
2137       \regex_replace_all:NnN
2138         \c_@@_columncolor_regex
2139         { \c { @@_columncolor_preamble } }
2140         \g_@@_array_preamble_tl
2141     }
2142   }
2143   {
2144     \cs_new_protected:Npn \@@_replace_columncolor:
2145     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2146   }
2147 }

```

```

2148 \cs_new_protected:Npn \@@_transform_preamble_ii:
2149 {

```

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

```

2150   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2151   {
2152     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2153     { \bool_gset_true:N \g_@@_delims_bool }
2154   }
2155   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2156   \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).

```

2157   \int_if_zero:nTF \l_@@_first_col_int
2158   { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2159   {

```

```

2160     \bool_if:NF \g_@@_delims_bool
2161     {
2162         \bool_if:NF \l_@@_tabular_bool
2163         {
2164             \clist_if_empty:NT \l_@@_vlines_clist
2165             {
2166                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2167                 { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2168             }
2169         }
2170     }
2171 }
2172 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2173 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2174 {
2175     \bool_if:NF \g_@@_delims_bool
2176     {
2177         \bool_if:NF \l_@@_tabular_bool
2178         {
2179             \clist_if_empty:NT \l_@@_vlines_clist
2180             {
2181                 \bool_if:NF \l_@@_exterior_arraycolsep_bool
2182                 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2183             }
2184         }
2185     }
2186 }

```

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`).

```

2187     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2188     {
2189         \tl_gput_right:Nn \g_@@_array_preamble_tl
2190         { > { \@@_error_too_much_cols: } 1 }
2191     }
2192 }

```

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.

```

2193 \cs_new_protected:Npn \@@_rec_preamble:n #1
2194 {

```

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

```

2195     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2196     { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2197     {

```

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

```

2198     \cs_if_exist:cTF { NC @ find @ #1 }
2199     {
2200         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2201         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2202     }
2203     {

```

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`.

```

2204         \str_if_eq:nnTF { #1 } { S }
2205         { \@@_fatal:n { unknown~column~type~S } }
2206         { \@@_fatal:nn { unknown~column~type } { #1 } }
2207     }
2208 }
2209 }

```

For c, l and r

```

2210 \cs_new:Npn \@@_c #1
2211 {
2212     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2213     \tl_gclear:N \g_@@_pre_cell_tl
2214     \tl_gput_right:Nn \g_@@_array_preamble_tl
2215     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2216     \int_gincr:N \c@jCol
2217     \@@_rec_preamble_after_col:n
2218 }

2219 \cs_new:Npn \@@_l #1
2220 {
2221     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2222     \tl_gclear:N \g_@@_pre_cell_tl
2223     \tl_gput_right:Nn \g_@@_array_preamble_tl
2224     {
2225         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2226         l
2227         < \@@_cell_end:
2228     }
2229     \int_gincr:N \c@jCol
2230     \@@_rec_preamble_after_col:n
2231 }

2232 \cs_new:Npn \@@_r #1
2233 {
2234     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2235     \tl_gclear:N \g_@@_pre_cell_tl
2236     \tl_gput_right:Nn \g_@@_array_preamble_tl
2237     {
2238         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2239         r
2240         < \@@_cell_end:
2241     }
2242     \int_gincr:N \c@jCol
2243     \@@_rec_preamble_after_col:n
2244 }

```

For ! and @

```

2245 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2246 {
2247     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2248     \@@_rec_preamble:n
2249 }
2250 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2251 \cs_new:cpn { @@ _ | } #1
2252 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2253     \int_incr:N \l_tmpa_int
2254     \@@_make_preamble_i_i:n
2255 }

```

```

2256 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2257 {
2258   \str_if_eq:nnTF { #1 } { | }
2259     { \use:c { @@ _ | } | }
2260     { \@@_make_preamble_i_ii:nn { } #1 }
2261 }
2262 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2263 {
2264   \str_if_eq:nnTF { #2 } { [ ] }
2265     { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2266     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2267 }
2268 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2269 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2270 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2271 {
2272   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2273   \tl_gput_right:Ne \g_@@_array_preamble_tl
2274   {

```

Here, the command `\dim_eval:n` is mandatory.

```

2275     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2276   }
2277   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2278   {
2279     \@@_vline:n
2280     {
2281       position = \int_eval:n { \c@jCol + 1 } ,
2282       multiplicity = \int_use:N \l_tmpa_int ,
2283       total-width = \dim_use:N \l_@@_rule_width_dim ,
2284       #2
2285     }

```

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.

```

2286   }
2287   \int_zero:N \l_tmpa_int
2288   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2289   \@@_rec_preamble:n #1
2290 }
2291 \cs_new:cpn { @@ _ > } #1 #2
2292 {
2293   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2294   \@@_rec_preamble:n
2295 }
2296 \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.

```

2297 \keys_define:nn { nicematrix / p-column }
2298 {
2299   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2300   r .value_forbidden:n = true ,
2301   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2302   c .value_forbidden:n = true ,
2303   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2304   l .value_forbidden:n = true ,
2305   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2306   S .value_forbidden:n = true ,
2307   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,

```

```

2308     p .value_forbidden:n = true ,
2309     t .meta:n = p ,
2310     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2311     m .value_forbidden:n = true ,
2312     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2313     b .value_forbidden:n = true
2314 }

```

For p but also b and m.

```

2315 \cs_new:Npn \@@_p #1
2316 {
2317   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2318   \@@_make_preamble_ii_i:n
2319 }
2320 \cs_set_eq:NN \@@_b \@@_p
2321 \cs_set_eq:NN \@@_m \@@_p
2322 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2323 {
2324   \str_if_eq:nnTF { #1 } { [ ]
2325     { \@@_make_preamble_ii_ii:w [ ]
2326       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2327     }
2328   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2329     { \@@_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.

```

2330 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2331 {

```

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).

```

2332   \str_set:Nn \l_@@_hpos_col_str { j }
2333   \@@_keys_p_column:n { #1 }
2334   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2335 }
2336 \cs_new_protected:Npn \@@_keys_p_column:n #1
2337 { \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.

```

2338 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2339 {
2340   \use:e
2341   {
2342     \@@_make_preamble_ii_v:nnnnnnn
2343     { \str_if_eq:nnTF \l_@@_vpos_col_str { p } { t } { b } }
2344     { \dim_eval:n { #1 } }
2345   }

```

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.

```

2346   \str_if_eq:nnTF \l_@@_hpos_col_str { j }
2347     { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2348   {

```

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

```

2349         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2350         { \str_lowercase:o \l_@@_hpos_col_str }
2351     }
2352     \IfPackageLoadedTF { ragged2e }
2353     {
2354         \str_case:on \l_@@_hpos_col_str
2355         {
2356             c { \exp_not:N \Centering }
2357             l { \exp_not:N \RaggedRight }
2358             r { \exp_not:N \RaggedLeft }
2359         }
2360     }
2361     {
2362         \str_case:on \l_@@_hpos_col_str
2363         {
2364             c { \exp_not:N \centering }
2365             l { \exp_not:N \raggedright }
2366             r { \exp_not:N \raggedleft }
2367         }
2368     }
2369     #3
2370 }
2371 { \str_if_eq:nnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2372 { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2373 { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2374 { #2 }
2375 {
2376     \str_case:onF \l_@@_hpos_col_str
2377     {
2378         { j } { c }
2379         { si } { c }
2380     }

```

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

```

2381         { \str_lowercase:o \l_@@_hpos_col_str }
2382     }
2383 }

```

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

```

2384     \int_gincr:N \c@jCol
2385     \@@_rec_preamble_after_col:n
2386 }

```

#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*.

```

2387 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2388 {
2389     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2390     {
2391         \tl_gput_right:Nn \g_@@_array_preamble_tl
2392         { > \@@_test_if_empty_for_S: }

```



```

2393     }
2394     {
2395         \tl_gput_right:Nn \g_@@_array_preamble_tl
2396         { > { \@@_test_if_empty: } }
2397     }
2398     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2399     \tl_gclear:N \g_@@_pre_cell_tl
2400     \tl_gput_right:Nn \g_@@_array_preamble_tl
2401     {
2402         > {

```

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.

```

2403         \dim_set:Nn \l_@@_col_width_dim { #2 }
2404         \bool_if:NT \c_@@_testphase_table_bool
2405         { \tag_struct_begin:n { tag = Div } }
2406         \@@_cell_begin:w

```

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

```

2407         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2408         \everypar
2409         {
2410             \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2411             \everypar { }
2412         }
2413         \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.).

```

2414         #3

```

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

```

2415         \g_@@_row_style_tl
2416         \arraybackslash
2417         #5
2418     }
2419     #8
2420     < {
2421         #6

```

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

```

2422         \@finalstrut \@arstrutbox
2423         \use:c { end #7 }

```

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

```

2424         #4
2425         \@@_cell_end:
2426         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2427     }
2428 }
2429 }

```

```

2430 \cs_new_protected:Npn \@@_test_if_empty:
2431 { \peek_after:Nw \@@_test_if_empty_i: }
2432
2433 \cs_new_protected:Npn \@@_test_if_empty_i:
2434 { \peek_meaning_remove:NT \ignorespaces { \@@_test_if_empty_ii: } }
2435
2436 \cs_new_protected:Npn \@@_test_if_empty_ii:
2437 { \peek_after:Nw \@@_test_if_empty_iii: }

```

```

2438 \bool_if:NTF \c_@@_tagging_array_bool
2439 {
2440   \cs_new_protected:Npn \@@_test_if_empty_iii:
2441     { \peek_meaning:NTF \textonly@unskip \@@_nullify_cell: \ignorespaces }
2442 }

```

In the old version of `array`, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `colcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty... First, we test if the next token is `\ignorespaces` and it's not very easy...

```

2443 {
2444   \cs_new_protected:Npn \@@_test_if_empty_iii:
2445     { \peek_meaning:NTF \unskip \@@_nullify_cell: \ignorespaces }
2446 }

```

```

2447 \cs_new_protected:Npn \@@_nullify_cell:
2448 {
2449   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2450   {
2451     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2452     \skip_horizontal:N \l_@@_col_width_dim
2453   }
2454 }
2455 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2456 {
2457   \peek_meaning:NT \_siunitx_table_skip:n
2458   {
2459     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2460     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2461   }
2462 }

```

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.

```

2463 \cs_new_protected:Npn \@@_center_cell_box:
2464 {

```

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

```

2465   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2466   {
2467     \int_compare:nNnT
2468       { \box_ht:N \l_@@_cell_box }
2469     >

```

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*).

```

2470     { \box_ht:N \strutbox }
2471   {
2472     \hbox_set:Nn \l_@@_cell_box
2473     {
2474       \box_move_down:nn
2475       {
2476         ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2477           + \baselineskip ) / 2
2478       }
2479       { \box_use:N \l_@@_cell_box }
2480     }

```

```

2481     }
2482   }
2483 }

```

For V (similar to the V of varwidth).

```

2484 \cs_new:Npn \@@_V #1 #2
2485 {
2486   \str_if_eq:nnTF { #1 } { [ ]
2487     { \@@_make_preamble_V_i:w [ ]
2488       { \@@_make_preamble_V_i:w [ ] { #2 } }
2489     }
2490 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2491   { \@@_make_preamble_V_ii:nn { #1 } }
2492 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2493 {
2494   \str_set:Nn \l_@@_vpos_col_str { p }
2495   \str_set:Nn \l_@@_hpos_col_str { j }
2496   \@@_keys_p_column:n { #1 }
2497   \IfPackageLoadedTF { varwidth }
2498     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2499     {
2500       \@@_error_or_warning:n { varwidth-not-loaded }
2501       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2502     }
2503 }

```

For w and W

```

2504 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2505 \cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

#1 is a special argument: empty for w and equal to \@@_special_W: for W;

#2 is the type of column (w or W);

#3 is the type of horizontal alignment (c, l, r or s);

#4 is the width of the column.

```

2506 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2507 {
2508   \str_if_eq:nnTF { #3 } { s }
2509     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2510     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2511 }

```

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.

```

2512 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2513 {
2514   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2515   \tl_gclear:N \g_@@_pre_cell_tl
2516   \tl_gput_right:Nn \g_@@_array_preamble_tl
2517     {
2518       > {
2519         \dim_set:Nn \l_@@_col_width_dim { #2 }
2520         \@@_cell_begin:w
2521         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2522       }
2523       c
2524       < {
2525         \@@_cell_end_for_w_s:
2526         #1
2527         \@@_adjust_size_box:
2528         \box_use_drop:N \l_@@_cell_box
2529       }

```

```

2530     }
2531     \int_gincr:N \c@jCol
2532     \@@_rec_preamble_after_col:n
2533 }

```

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

```

2534 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2535 {
2536   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2537   \tl_gc clear:N \g_@@_pre_cell_tl
2538   \tl_gput_right:Nn \g_@@_array_preamble_tl
2539   {
2540     > {

```

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.

```

2541       \dim_set:Nn \l_@@_col_width_dim { #4 }
2542       \hbox_set:Nw \l_@@_cell_box
2543       \@@_cell_begin:w
2544       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2545     }
2546     c
2547     < {
2548       \@@_cell_end:
2549       \hbox_set_end:
2550       #1
2551       \@@_adjust_size_box:
2552       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2553     }
2554   }

```

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

```

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

2558 \cs_new_protected:Npn \@@_special_W:
2559 {
2560   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2561   { \@@_warning:n { W~warning } }
2562 }

```

For S (of siunitx).

```

2563 \cs_new:Npn \@@_S #1 #2
2564 {
2565   \str_if_eq:nnTF { #1 } { [ ] }
2566   { \@@_make_preamble_S:w [ ] }
2567   { \@@_make_preamble_S:w [ ] { #2 } }
2568 }

2569 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2570 { \@@_make_preamble_S_i:n { #1 } }

2571 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2572 {
2573   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2574   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2575   \tl_gc clear:N \g_@@_pre_cell_tl
2576   \tl_gput_right:Nn \g_@@_array_preamble_tl
2577   {
2578     > {
2579       \@@_cell_begin:w
2580       \keys_set:nn { siunitx } { #1 }
2581       \siunitx_cell_begin:w

```

```

2582     }
2583     c
2584     < { \siunitx_cell_end: \@@_cell_end: }
2585 }

```

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

```

2586     \int_gincr:N \c@jCol
2587     \@@_rec_preamble_after_col:n
2588 }

```

For (, [and \{.

```

2589 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2590 {
2591     \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.

```

2592     \int_if_zero:nTF \c@jCol
2593     {
2594         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2595         {

```

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

```

2596             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2597             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2598             \@@_rec_preamble:n #2
2599         }
2600     {
2601         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2602         \@@_make_preamble_iv:nn { #1 } { #2 }
2603     }
2604 }
2605 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2606 }
2607 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2608 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2609 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2610 {
2611     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2612     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2613     \tl_if_in:nnTF { ( [ \{ ) ] \} } \left \right } { #2 }
2614     {
2615         \@@_error:nn { delimiter~after~opening } { #2 }
2616         \@@_rec_preamble:n
2617     }
2618     { \@@_rec_preamble:n #2 }
2619 }

```

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

```

2620 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2621 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2622 {
2623     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2624     \tl_if_in:nnTF { ) ] \} } { #2 }
2625     { \@@_make_preamble_v:nnn #1 #2 }
2626     {
2627         \str_if_eq:nnTF { \@@_stop: } { #2 }
2628         {

```

```

2629     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2630     { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2631     {
2632       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2633       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2634       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2635       \@@_rec_preamble:n #2
2636     }
2637   }
2638   {
2639     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2640     { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } } }
2641     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2642     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2643     \@@_rec_preamble:n #2
2644   }
2645 }
2646 }
2647 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2648 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2649 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2650 {
2651   \str_if_eq:nnTF { \@@_stop: } { #3 }
2652   {
2653     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2654     {
2655       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2656       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2657       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2658       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2659     }
2660     {
2661       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2662       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2663       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2664       \@@_error:nn { double~closing~delimiter } { #2 }
2665     }
2666   }
2667   {
2668     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2669     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2670     \@@_error:nn { double~closing~delimiter } { #2 }
2671     \@@_rec_preamble:n #3
2672   }
2673 }
2674 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2675 { \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 @{. . .}.

```

2676 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2677 {
2678   \str_if_eq:nnTF { #1 } { < }
2679   \@@_rec_preamble_after_col_i:n
2680   {
2681     \str_if_eq:nnTF { #1 } { @ }
2682     \@@_rec_preamble_after_col_ii:n
2683     {
2684       \str_if_eq:nnTF \l_@@_vlines_clist { all }
2685       {

```

```

2686         \tl_gput_right:Nn \g_@@_array_preamble_tl
2687         { ! { \skip_horizontal:N \arrayrulewidth } }
2688     }
2689     {
2690         \clist_if_in:NeT \l_@@_vlines_clist
2691         { \int_eval:n { \c@jCol + 1 } }
2692         {
2693             \tl_gput_right:Nn \g_@@_array_preamble_tl
2694             { ! { \skip_horizontal:N \arrayrulewidth } }
2695         }
2696     }
2697     \@@_rec_preamble:n { #1 }
2698 }
2699 }
2700 }

2701 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2702 {
2703     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2704     \@@_rec_preamble_after_col:n
2705 }

```

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.

```

2706 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2707 {
2708     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2709     {
2710         \tl_gput_right:Nn \g_@@_array_preamble_tl
2711         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2712     }
2713     {
2714         \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2715         {
2716             \tl_gput_right:Nn \g_@@_array_preamble_tl
2717             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2718         }
2719         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2720     }
2721     \@@_rec_preamble:n
2722 }

2723 \cs_new:cpn { @@ _ * } #1 #2 #3
2724 {
2725     \tl_clear:N \l_tmpa_tl
2726     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2727     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2728 }

```

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.

```

2729 \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }

```

For the case of a letter X. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a [after the letter X.

```

2730 \cs_new:Npn \@@_X #1 #2
2731 {
2732     \str_if_eq:nnTF { #2 } { [ ]
2733     { \@@_make_preamble_X:w [ ] }
2734     { \@@_make_preamble_X:w [ ] #2 }
2735 }

```

```

2736 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2737 { \@@_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).

```

2738 \keys_define:nn { nicematrix / X-column }
2739 { 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.

```

2740 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2741 {

```

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).

```

2742 \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).

```

2743 \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*.

```

2744 \int_zero_new:N \l_@@_weight_int
2745 \int_set_eq:NN \l_@@_weight_int \c_one_int
2746 \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l_tmpa_tl

```

2747 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2748 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2749 {
2750 \@@_error_or_warning:n { negative-weight }
2751 \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2752 }
2753 \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).

```

2754 \bool_if:NTF \l_@@_X_columns_aux_bool
2755 {
2756 \@@_make_preamble_ii_iv:nnn
2757 { \l_@@_weight_int \l_@@_X_columns_dim }
2758 { minipage }
2759 { \@@_no_update_width: }
2760 }
2761 {
2762 \tl_gput_right:Nn \g_@@_array_preamble_tl
2763 {
2764 > {
2765 \@@_cell_begin:w
2766 \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.

```

2767 \NotEmpty

```

The following code will nullify the box of the cell.

```

2768 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2769 { \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`.

```

2770         \begin { minipage } { 5 cm } \arraybackslash
2771     }
2772     c
2773     < {
2774         \end { minipage }
2775         \@@_cell_end:
2776     }
2777 }
2778 \int_gincr:N \c@jCol
2779 \@@_rec_preamble_after_col:n
2780 }
2781 }

2782 \cs_new_protected:Npn \@@_no_update_width:
2783 {
2784     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2785     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2786 }

```

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

```

2787 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2788 {
2789     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2790     { \int_eval:n { \c@jCol + 1 } }
2791     \tl_gput_right:Ne \g_@@_array_preamble_tl
2792     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2793     \@@_rec_preamble:n
2794 }

```

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.

```

2795 \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).

```

2796 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2797 { \@@_fatal:n { Preamble-forgotten } }
2798 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2799 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2800 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```

2801 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2802 {

```

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`.

```

2803     \multispan { #1 }
2804     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2805     \begingroup
2806     \bool_if:NT \c_@@_testphase_table_bool
2807     { \tbl_update_multicolumn_cell_data:n { #1 } }
2808     \cs_set_nopar:Npn \@addamp
2809     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```
2810 \tl_gclear:N \g_@@_preamble_tl
2811 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2812 \exp_args:No \mkpream \g_@@_preamble_tl
2813 \@addtopreamble \empty
2814 \endgroup
2815 \bool_if:NT \c_@@_testphase_table_bool
2816 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2817 \int_compare:nNtT { #1 } > \c_one_int
2818 {
2819   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2820   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2821   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2822   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2823   {
2824     {
2825       \int_if_zero:nTF \c@jCol
2826       { \int_eval:n { \c@iRow + 1 } }
2827       { \int_use:N \c@iRow }
2828     }
2829     { \int_eval:n { \c@jCol + 1 } }
2830     {
2831       \int_if_zero:nTF \c@jCol
2832       { \int_eval:n { \c@iRow + 1 } }
2833       { \int_use:N \c@iRow }
2834     }
2835     { \int_eval:n { \c@jCol + #1 } }
2836     { } % for the name of the block
2837   }
2838 }
```

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

```
2839 \RenewDocumentCommand \cellcolor { 0 { } m }
2840 {
2841   \@@_test_color_inside:
2842   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2843   {
2844     \@@_rectanglecolor [ ##1 ]
2845     { \exp_not:n { ##2 } }
2846     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2847     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2848   }
2849   \ignorespaces
2850 }
```

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

```
2851 \cs_set_nopar:Npn \@sharp { #3 }
2852 \@arstrut
2853 \@preamble
2854 \null
```

We add some lines.

```
2855 \int_gadd:Nn \c@jCol { #1 - 1 }
2856 \int_compare:nNtT \c@jCol > \g_@@_col_total_int
2857 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2858 \ignorespaces
2859 }
```

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`.

```

2860 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2861 {
2862   \str_case:nnF { #1 }
2863   {
2864     c { \@@_make_m_preamble_i:n #1 }
2865     l { \@@_make_m_preamble_i:n #1 }
2866     r { \@@_make_m_preamble_i:n #1 }
2867     > { \@@_make_m_preamble_ii:nn #1 }
2868     ! { \@@_make_m_preamble_ii:nn #1 }
2869     @ { \@@_make_m_preamble_ii:nn #1 }
2870     | { \@@_make_m_preamble_iii:n #1 }
2871     p { \@@_make_m_preamble_iv:nnn t #1 }
2872     m { \@@_make_m_preamble_iv:nnn c #1 }
2873     b { \@@_make_m_preamble_iv:nnn b #1 }
2874     w { \@@_make_m_preamble_v:nnnn { } #1 }
2875     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2876     \q_stop { }
2877   }
2878   {
2879     \cs_if_exist:cTF { NC @ find @ #1 }
2880     {
2881       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2882       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2883     }
2884     {
2885       \str_if_eq:nnTF { #1 } { S }
2886       { \@@_fatal:n { unknown~column~type~S } }
2887       { \@@_fatal:nn { unknown~column~type } { #1 } }
2888     }
2889   }
2890 }

```

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

```

2891 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2892 {
2893   \tl_gput_right:Nn \g_@@_preamble_tl
2894   {
2895     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2896     #1
2897     < \@@_cell_end:
2898   }

```

We test for the presence of a `<`.

```

2899   \@@_make_m_preamble_x:n
2900 }

```

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

```

2901 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2902 {
2903   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2904   \@@_make_m_preamble:n
2905 }

```

For `|`

```

2906 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2907 {
2908   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2909   \@@_make_m_preamble:n
2910 }

```

For p, m and b

```

2911 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2912 {
2913   \tl_gput_right:Nn \g_@@_preamble_tl
2914   {
2915     > {
2916       \@@_cell_begin:w
2917       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2918       \mode_leave_vertical:
2919       \arraybackslash
2920       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2921     }
2922     c
2923     < {
2924       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2925       \end { minipage }
2926       \@@_cell_end:
2927     }
2928   }

```

We test for the presence of a <.

```

2929   \@@_make_m_preamble_x:n
2930 }

```

For w and W

```

2931 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2932 {
2933   \tl_gput_right:Nn \g_@@_preamble_tl
2934   {
2935     > {
2936       \dim_set:Nn \l_@@_col_width_dim { #4 }
2937       \hbox_set:Nw \l_@@_cell_box
2938       \@@_cell_begin:w
2939       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2940     }
2941     c
2942     < {
2943       \@@_cell_end:
2944       \hbox_set_end:
2945       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2946       #1
2947       \@@_adjust_size_box:
2948       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2949     }
2950   }

```

We test for the presence of a <.

```

2951   \@@_make_m_preamble_x:n
2952 }

```

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

```

2953 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2954 {
2955   \str_if_eq:nnTF { #1 } { < }
2956   \@@_make_m_preamble_ix:n
2957   { \@@_make_m_preamble:n { #1 } }
2958 }
2959 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2960 {
2961   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2962   \@@_make_m_preamble_x:n
2963 }

```

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`).

```

2964 \cs_new_protected:Npn \@@_put_box_in_flow:
2965 {
2966   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2967   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2968   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2969     { \box_use_drop:N \l_tmpa_box }
2970     \@@_put_box_in_flow_i:
2971 }

```

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

```

2972 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2973 {
2974   \pgfpicture
2975     \@@_qpoint:n { row - 1 }
2976     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2977     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2978     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2979     \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).

```

2980   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2981     {
2982       \int_set:Nn \l_tmpa_int
2983         {
2984           \str_range:Nnn
2985             \l_@@_baseline_tl
2986             6
2987             { \tl_count:o \l_@@_baseline_tl }
2988         }
2989       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2990     }
2991     {
2992       \str_if_eq:onTF \l_@@_baseline_tl { t }
2993       { \int_set_eq:NN \l_tmpa_int \c_one_int }
2994       {
2995         \str_if_eq:onTF \l_@@_baseline_tl { b }
2996         { \int_set_eq:NN \l_tmpa_int \c@iRow }
2997         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2998       }
2999       \bool_lazy_or:nnT
3000         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3001         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3002       {
3003         \@@_error:n { bad-value-for-baseline }
3004         \int_set_eq:NN \l_tmpa_int \c_one_int
3005       }
3006       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3007       \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3008     }
3009     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3010   \endpgfpicture
3011   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3012   \box_use_drop:N \l_tmpa_box
3013 }

```

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).

```
3014 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3015 {
```

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.

```
3016   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3017   {
3018     \int_compare:nNnT \c_jCol > \c_one_int
3019     {
3020       \box_set_wd:Nn \l_@@_the_array_box
3021       { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3022     }
3023   }
```

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

```
3024   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3025   \bool_if:NT \l_@@_caption_above_bool
3026   {
3027     \tl_if_empty:NF \l_@@_caption_tl
3028     {
3029       \bool_set_false:N \g_@@_caption_finished_bool
3030       \int_gzero:N \c@tabularnote
3031       \@@_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).

```
3032       \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3033       {
3034         \tl_gput_right:Ne \g_@@_aux_tl
3035         {
3036           \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3037           { \int_use:N \g_@@_notes_caption_int }
3038         }
3039         \int_gzero:N \g_@@_notes_caption_int
3040       }
3041     }
3042   }
```

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

```
3043   \hbox
3044   {
3045     \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.

```
3046     \@@_create_extra_nodes:
3047     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3048   }
```

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

```
3049   \bool_lazy_any:nT
3050   {
3051     { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

3052     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3053     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3054   }
3055   \@@_insert_tabularnotes:
3056   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3057   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3058   \end { minipage }
3059 }

```

```

3060 \cs_new_protected:Npn \@@_insert_caption:
3061 {
3062   \tl_if_empty:NF \l_@@_caption_tl
3063   {
3064     \cs_if_exist:NTF \@cptype
3065     { \@@_insert_caption_i: }
3066     { \@@_error:n { caption~outside~float } }
3067   }
3068 }

```

```

3069 \cs_new_protected:Npn \@@_insert_caption_i:
3070 {
3071   \group_begin:

```

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

```

3072   \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.

```

3073   \IfPackageLoadedT { floatrow }
3074   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3075   \tl_if_empty:NTF \l_@@_short_caption_tl
3076   { \caption }
3077   { \caption [ \l_@@_short_caption_tl ] }
3078   { \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.

```

3079   \bool_if:NF \g_@@_caption_finished_bool
3080   {
3081     \bool_gset_true:N \g_@@_caption_finished_bool
3082     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3083     \int_gzero:N \c@tabularnote
3084   }
3085   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3086   \group_end:
3087 }

```

```

3088 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3089 {
3090   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3091   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3092 }

```

```

3093 \cs_new_protected:Npn \@@_insert_tabularnotes:
3094 {
3095   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3096   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3097   \skip_vertical:N 0.65ex

```

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

```

3098   \group_begin:
3099   \l_@@_notes_code_before_tl
3100   \tl_if_empty:NF \g_@@_tabularnote_tl
3101   {
3102     \g_@@_tabularnote_tl \par
3103     \tl_gclear:N \g_@@_tabularnote_tl
3104   }

```

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.

```

3105   \int_compare:nNnT \c@tabularnote > \c_zero_int
3106   {
3107     \bool_if:NTF \l_@@_notes_para_bool
3108     {
3109       \begin { tabularnotes* }
3110         \seq_map_inline:Nn \g_@@_notes_seq
3111         { \@@_one_tabularnote:nm ##1 }
3112         \strut
3113       \end { tabularnotes* }

```

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

```

3114     \par
3115   }
3116   {
3117     \tabularnotes
3118     \seq_map_inline:Nn \g_@@_notes_seq
3119     { \@@_one_tabularnote:nm ##1 }
3120     \strut
3121     \endtabularnotes
3122   }
3123 }
3124 \unskip
3125 \group_end:
3126 \bool_if:NT \l_@@_notes_bottomrule_bool
3127 {
3128   \IfPackageLoadedTF { booktabs }
3129   {

```

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

```

3130     \skip_vertical:N \aboverulesep

```

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

```

3131     { \CT@arc@ \hrule height \heavyrulewidth }
3132   }
3133   { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3134 }
3135 \l_@@_notes_code_after_tl
3136 \seq_gclear:N \g_@@_notes_seq
3137 \seq_gclear:N \g_@@_notes_in_caption_seq
3138 \int_gzero:N \c@tabularnote
3139 }

```

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

```

3140 \cs_set_protected:Npn \@@_one_tabularnote:nm #1
3141 {
3142   \tl_if_novalue:nTF { #1 }
3143   { \item }
3144   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3145 }

```


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`.

```

3146 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3147 {
3148   \pgfpicture
3149     \@@_qpoint:n { row - 1 }
3150     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3151     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3152     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3153   \endpgfpicture
3154   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3155   \int_if_zero:nT \l_@@_first_row_int
3156     {
3157       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3158       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3159     }
3160   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3161 }

```

Now, the general case.

```

3162 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3163 {

```

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

```

3164   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3165   { \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`.

```

3166   \pgfpicture
3167   \@@_qpoint:n { row - 1 }
3168   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3169   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3170     {
3171       \int_set:Nn \l_tmpa_int
3172         {
3173           \str_range:Nnn
3174             \l_@@_baseline_tl
3175             6
3176             { \tl_count:o \l_@@_baseline_tl }
3177         }
3178       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3179     }
3180   {
3181     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3182     \bool_lazy_or:nnT
3183       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3184       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3185     {
3186       \@@_error:n { bad-value-for-baseline }
3187       \int_set:Nn \l_tmpa_int 1
3188     }
3189     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3190   }
3191   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3192   \endpgfpicture
3193   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3194   \int_if_zero:nT \l_@@_first_row_int
3195     {
3196       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3197       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3198     }
3199   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3200 }
```

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.

```
3201 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3202 {
```

We will compute the real width of both delimiters used.

```
3203   \dim_zero_new:N \l_@@_real_left_delim_dim
3204   \dim_zero_new:N \l_@@_real_right_delim_dim
3205   \hbox_set:Nn \l_tmpb_box
3206   {
3207     \c_math_toggle_token
3208     \left #1
3209     \vcenter
3210     {
3211       \vbox_to_ht:nn
3212       { \box_ht_plus_dp:N \l_tmpa_box }
3213       { }
3214     }
3215     \right .
3216     \c_math_toggle_token
3217   }
3218   \dim_set:Nn \l_@@_real_left_delim_dim
3219   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3220   \hbox_set:Nn \l_tmpb_box
3221   {
3222     \c_math_toggle_token
3223     \left .
3224     \vbox_to_ht:nn
3225     { \box_ht_plus_dp:N \l_tmpa_box }
3226     { }
3227     \right #2
3228     \c_math_toggle_token
3229   }
3230   \dim_set:Nn \l_@@_real_right_delim_dim
3231   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3232   \skip_horizontal:N \l_@@_left_delim_dim
3233   \skip_horizontal:N -\l_@@_real_left_delim_dim
3234   \@@_put_box_in_flow:
3235   \skip_horizontal:N \l_@@_right_delim_dim
3236   \skip_horizontal:N -\l_@@_real_right_delim_dim
3237 }
```

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).

```
3238 \NewDocumentEnvironment { @@-normal-syntax } { }
```

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it's only a security). In order to detect whether it is empty, we test whether the next token is `\end` and, if it's the case, we test if this is the end of the environment (if it is not, an standard error will be raised by LaTeX for incorrect nested environments).

```
3239 {
3240   \peek_remove_spaces:n
3241   {
3242     \peek_meaning:NTF \end
3243     \@@_analyze_end:Nn
```

```

3244     {
3245     \@@_transform_preamble:

```

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

```

3246     \@@_array:o \g_@@_array_preamble_tl
3247     }
3248   }
3249 }
3250 {
3251 \@@_create_col_nodes:
3252 \endarray
3253 }

```

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

```

3254 \NewDocumentEnvironment { @@-light-syntax } { b }
3255 {

```

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`.

```

3256 \tl_if_empty:nT { #1 }
3257   { \@@_fatal:n { empty-environment } }
3258 \tl_if_in:nnT { #1 } { & }
3259   { \@@_fatal:n { ampersand-in~light-syntax } }
3260 \tl_if_in:nnT { #1 } { \ }
3261   { \@@_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`.

```

3262 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3263   }

```

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.

```

3264   {
3265     \@@_create_col_nodes:
3266     \endarray
3267   }

3268 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3269   {
3270     \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).

```

3271     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3272     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3273     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3274       \seq_set_split:Nee
3275       \seq_set_split:Non
3276     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3277     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3278     \tl_if_empty:NF \l_tmpa_tl
3279     { \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.

```
3280   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3281   { \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`).

```
3282   \tl_build_begin:N \l_@@_new_body_tl
3283   \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3284   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3285   \@@_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).

```
3286   \seq_map_inline:Nn \l_@@_rows_seq
3287   {
3288     \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3289     \@@_line_with_light_syntax:n { #1 }
3290   }
3291   \tl_build_end:N \l_@@_new_body_tl
3292   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3293   {
3294     \int_set:Nn \l_@@_last_col_int
3295     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3296   }
```

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.

```
3297   \@@_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`).

```
3298   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3299   }
3300   \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3301   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3302   {
3303     \seq_clear_new:N \l_@@_cells_seq
3304     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3305     \int_set:Nn \l_@@_nb_cols_int
3306     {
3307       \int_max:nn
3308       \l_@@_nb_cols_int
3309       { \seq_count:N \l_@@_cells_seq }
3310     }
3311     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3312     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3313     \seq_map_inline:Nn \l_@@_cells_seq
3314     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3315   }
```

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`.

```
3316   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3317   {
3318     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3319     { \@@_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.

```
3320   \end { #2 }
3321 }
```

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`).

```
3322 \cs_new:Npn \@@_create_col_nodes:
3323 {
3324   \crr
3325   \int_if_zero:nT \l_@@_first_col_int
3326   {
3327     \omit
3328     \hbox_overlap_left:n
3329     {
3330       \bool_if:NT \l_@@_code_before_bool
3331       { \pgfsys@markposition { \@@_env: - col - 0 } }
3332       \pgfpicture
3333       \pgfrememberpicturepositiononpagetrue
3334       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3335       \str_if_empty:NF \l_@@_name_str
3336       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3337       \endpgfpicture
3338       \skip_horizontal:N 2\col@sep
3339       \skip_horizontal:N \g_@@_width_first_col_dim
3340     }
3341     &
3342   }
3343   \omit
```

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

```
3344   \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`).

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

```

3372     }
3373   }
3374   \pgfpicture
3375   \pgfrememberpicturepositiononpagetrue
3376   \pgfcoordinate { \@@_env: - col - 1 }
3377   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3378   \str_if_empty:NF \l_@@_name_str
3379   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3380   \endpgfpicture
3381 }

```

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.

```

3382   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3383   \bool_if:NF \l_@@_auto_columns_width_bool
3384   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3385   {
3386     \bool_lazy_and:nnTF
3387     \l_@@_auto_columns_width_bool
3388     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3389     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3390     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3391     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3392   }
3393   \skip_horizontal:N \g_tmpa_skip
3394   \hbox
3395   {
3396     \bool_if:NT \l_@@_code_before_bool
3397     {
3398       \hbox
3399       {
3400         \skip_horizontal:N -0.5\arrayrulewidth
3401         \pgfsys@markposition { \@@_env: - col - 2 }
3402         \skip_horizontal:N 0.5\arrayrulewidth
3403       }
3404     }
3405     \pgfpicture
3406     \pgfrememberpicturepositiononpagetrue
3407     \pgfcoordinate { \@@_env: - col - 2 }
3408     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3409     \str_if_empty:NF \l_@@_name_str
3410     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3411     \endpgfpicture
3412   }

```

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.

```

3413   \int_gset_eq:NN \g_tmpa_int \c_one_int
3414   \bool_if:NTF \g_@@_last_col_found_bool
3415   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3416   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3417   {
3418     &
3419     \omit
3420     \int_gincr:N \g_tmpa_int

```

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

```

3421     \skip_horizontal:N \g_tmpa_skip
3422     \bool_if:NT \l_@@_code_before_bool
3423     {

```

```

3424     \hbox
3425     {
3426         \skip_horizontal:N -0.5\arrayrulewidth
3427         \pgfsys@markposition
3428         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3429         \skip_horizontal:N 0.5\arrayrulewidth
3430     }
3431 }

```

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

```

3432     \pgfpicture
3433     \pgfrememberpicturepositiononpagetrue
3434     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3435     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3436     \str_if_empty:NF \l_@@_name_str
3437     {
3438         \pgfnodealias
3439         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3440         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3441     }
3442     \endpgfpicture
3443 }

```

```

3444 &
3445 \omit

```

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

```

3446     \int_if_zero:nT \g_@@_col_total_int
3447     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3448     \skip_horizontal:N \g_tmpa_skip
3449     \int_gincr:N \g_tmpa_int
3450     \bool_lazy_any:nF
3451     {
3452         \g_@@_delims_bool
3453         \l_@@_tabular_bool
3454         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3455         \l_@@_exterior_arraycolsep_bool
3456         \l_@@_bar_at_end_of_pream_bool
3457     }
3458     { \skip_horizontal:N -\col@sep }
3459     \bool_if:NT \l_@@_code_before_bool
3460     {
3461         \hbox
3462         {
3463             \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.

```

3464         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3465         { \skip_horizontal:N -\arraycolsep }
3466         \pgfsys@markposition
3467         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3468         \skip_horizontal:N 0.5\arrayrulewidth
3469         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3470         { \skip_horizontal:N \arraycolsep }
3471     }
3472 }
3473 \pgfpicture
3474 \pgfrememberpicturepositiononpagetrue
3475 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3476 {
3477     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3478         {
3479             \pgfpoint
3480             { - 0.5 \arrayrulewidth - \arraycolsep }
3481             \c_zero_dim
3482         }
3483         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3484     }
3485     \str_if_empty:NF \l_@@_name_str
3486     {
3487         \pgfnodealias
3488         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3489         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3490     }
3491     \endpgfpicture

3492     \bool_if:NT \g_@@_last_col_found_bool
3493     {
3494         \hbox_overlap_right:n
3495         {
3496             \skip_horizontal:N \g_@@_width_last_col_dim
3497             \skip_horizontal:N \col@sep
3498             \bool_if:NT \l_@@_code_before_bool
3499             {
3500                 \pgfsys@markposition
3501                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3502             }
3503             \pgfpicture
3504             \pgfrememberpicturepositiononpagetrue
3505             \pgfcoordinate
3506             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3507             \pgfpointorigin
3508             \str_if_empty:NF \l_@@_name_str
3509             {
3510                 \pgfnodealias
3511                 {
3512                     \l_@@_name_str - col
3513                     - \int_eval:n { \g_@@_col_total_int + 1 }
3514                 }
3515                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3516             }
3517             \endpgfpicture
3518         }
3519     }
3520     % \cr
3521 }

```

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

```

3522 \tl_const:Nn \c_@@_preamble_first_col_tl
3523 {
3524     >
3525     {

```

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

```

3526         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3527         \bool_gset_true:N \g_@@_after_col_zero_bool
3528         \@@_begin_of_row:

```

The contents of the cell is constructed in the box `\l_@@_cell_box` because we have to compute some dimensions of this box.

```

3529         \hbox_set:Nw \l_@@_cell_box
3530         \@@_math_toggle:
3531         \@@_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”.

```

3532     \int_compare:nNnT \c@iRow > \c_zero_int
3533     {
3534         \bool_lazy_or:nnT
3535         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3536         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3537         {
3538             \l_@@_code_for_first_col_tl
3539             \xglobal \colorlet { nicematrix-first-col } { . }
3540         }
3541     }
3542 }

```

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`.

```

3543     l
3544     <
3545     {
3546         \@@_math_toggle:
3547         \hbox_set_end:
3548         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3549         \@@_adjust_size_box:
3550         \@@_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.

```

3551     \dim_gset:Nn \g_@@_width_first_col_dim
3552     { \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.

```

3553     \hbox_overlap_left:n
3554     {
3555         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3556         \@@_node_for_cell:
3557         { \box_use_drop:N \l_@@_cell_box }
3558         \skip_horizontal:N \l_@@_left_delim_dim
3559         \skip_horizontal:N \l_@@_left_margin_dim
3560         \skip_horizontal:N \l_@@_extra_left_margin_dim
3561     }
3562     \bool_gset_false:N \g_@@_empty_cell_bool
3563     \skip_horizontal:N -2\col@sep
3564 }
3565 }

```

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

```

3566 \tl_const:Nn \c_@@_preamble_last_col_tl
3567 {
3568     >
3569     {
3570         \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).

```

3571     \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.

```

3572     \bool_gset_true:N \g_@@_last_col_found_bool
3573     \int_gincr:N \c@jCol
3574     \int_gset_eq:NN \g_@@_col_total_int \c@jCol

```

The contents of the cell is constructed in the box `\l_tmpa_box` because we have to compute some dimensions of this box.

```

3575     \hbox_set:Nw \l_@@_cell_box
3576     \@@_math_toggle:
3577     \@@_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”.

```

3578     \int_compare:nNnT \c@iRow > \c_zero_int
3579     {
3580         \bool_lazy_or:nnT
3581         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3582         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3583         {
3584             \l_@@_code_for_last_col_tl
3585             \xglobal \colorlet { nicematrix-last-col } { . }
3586         }
3587     }
3588 }
3589 l
3590 <
3591 {
3592     \@@_math_toggle:
3593     \hbox_set_end:
3594     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3595     \@@_adjust_size_box:
3596     \@@_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.

```

3597     \dim_gset:Nn \g_@@_width_last_col_dim
3598     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3599     \skip_horizontal:N -2\col@sep

```

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

```

3600     \hbox_overlap_right:n
3601     {
3602         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3603         {
3604             \skip_horizontal:N \l_@@_right_delim_dim
3605             \skip_horizontal:N \l_@@_right_margin_dim
3606             \skip_horizontal:N \l_@@_extra_right_margin_dim
3607             \@@_node_for_cell:
3608         }
3609     }
3610     \bool_gset_false:N \g_@@_empty_cell_bool
3611 }
3612 }

```

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

```

3613 \NewDocumentEnvironment { NiceArray } { }
3614 {
3615     \bool_gset_false:N \g_@@_delims_bool
3616     \str_if_empty:NT \g_@@_name_env_str
3617     { \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).

```

3618     \NiceArrayWithDelims . .
3619 }
3620 { \endNiceArrayWithDelims }

```

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

```

3621 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3622 {
3623     \NewDocumentEnvironment { #1 NiceArray } { }
3624     {

```

```

3625     \bool_gset_true:N \g_@@_delims_bool
3626     \str_if_empty:NT \g_@@_name_env_str
3627     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3628     \@@_test_if_math_mode:
3629     \NiceArrayWithDelims #2 #3
3630   }
3631   { \endNiceArrayWithDelims }
3632 }
3633 \@@_def_env:nnn p ( )
3634 \@@_def_env:nnn b [ ]
3635 \@@_def_env:nnn B \{ \}
3636 \@@_def_env:nnn v | |
3637 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3638 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3639 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3640 {
3641   \bool_set_false:N \l_@@_preamble_bool
3642   \tl_clear:N \l_tmpa_tl
3643   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3644     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3645   \tl_put_right:Nn \l_tmpa_tl
3646     {
3647     *
3648     {
3649       \int_case:nnF \l_@@_last_col_int
3650       {
3651         { -2 } { \c@MaxMatrixCols }
3652         { -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).

```

3653       }
3654       { \int_eval:n { \l_@@_last_col_int - 1 } }
3655     }
3656     { #2 }
3657   }
3658   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3659   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3660 }
3661 \clist_map_inline:nn { p , b , B , v , V }
3662 {
3663   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3664   {
3665     \bool_gset_true:N \g_@@_delims_bool
3666     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3667     \int_if_zero:nT \l_@@_last_col_int
3668     {
3669       \bool_set_true:N \l_@@_last_col_without_value_bool
3670       \int_set:Nn \l_@@_last_col_int { -1 }
3671     }
3672     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3673     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3674   }
3675   { \use:c { end #1 NiceArray } }
3676 }

```

We define also an environment `{NiceMatrix}`

```

3677 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3678 {
3679   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3680   \int_if_zero:nT \l_@@_last_col_int
3681   {
3682     \bool_set_true:N \l_@@_last_col_without_value_bool
3683     \int_set:Nn \l_@@_last_col_int { -1 }
3684   }
3685   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3686   \bool_lazy_or:nnT
3687     { \clist_if_empty_p:N \l_@@_vlines_clist }
3688     { \l_@@_except_borders_bool }
3689     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3690   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3691 }
3692 { \endNiceArray }

```

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

```

3693 \cs_new_protected:Npn \@@_NotEmpty:
3694 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3695 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3696 {

```

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

```

3697   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3698     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3699   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3700   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3701   \tl_if_empty:NF \l_@@_short_caption_tl
3702   {
3703     \tl_if_empty:NT \l_@@_caption_tl
3704     {
3705       \@@_error_or_warning:n { short-caption-without-caption }
3706       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3707     }
3708   }
3709   \tl_if_empty:NF \l_@@_label_tl
3710   {
3711     \tl_if_empty:NT \l_@@_caption_tl
3712     { \@@_error_or_warning:n { label-without-caption } }
3713   }
3714   \NewDocumentEnvironment { TabularNote } { b }
3715   {
3716     \bool_if:NTF \l_@@_in_code_after_bool
3717       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3718     {
3719       \tl_if_empty:NF \g_@@_tabularnote_tl
3720       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3721       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3722     }
3723   }
3724   { }
3725   \@@_settings_for_tabular:
3726   \NiceArray { #2 }
3727 }
3728 {
3729   \endNiceArray
3730   \bool_if:NT \c_@@_testphase_table_bool

```

```

3731     { \UseTaggingSocket { tbl / hmode / end } }
3732 }
3733 \cs_new_protected:Npn \@@_settings_for_tabular:
3734 {
3735   \bool_set_true:N \l_@@_tabular_bool
3736   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3737   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3738   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3739 }

3740 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3741 {
3742   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3743   \dim_zero_new:N \l_@@_width_dim
3744   \dim_set:Nn \l_@@_width_dim { #1 }
3745   \keys_set:nm { nicematrix / NiceTabular } { #2 , #4 }
3746   \@@_settings_for_tabular:
3747   \NiceArray { #3 }
3748 }
3749 {
3750   \endNiceArray
3751   \int_if_zero:nT \g_@@_total_X_weight_int
3752     { \@@_error:n { NiceTabularX~without~X } }
3753 }

3754 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3755 {
3756   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3757   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3758   \keys_set:nm { nicematrix / NiceTabular } { #2 , #4 }
3759   \@@_settings_for_tabular:
3760   \NiceArray { #3 }
3761 }
3762 { \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`).

```

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

```

```

3782     }
3783     { 1-1 }
3784     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3785   }
3786 }
3787 }

```

```

3788 \cs_new_protected:Npn \@@_after_array:
3789 {

```

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`.

```

3790     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3791     \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.

```

3792     \bool_if:NT \g_@@_last_col_found_bool
3793     { \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`.

```

3794     \bool_if:NT \l_@@_last_col_without_value_bool
3795     { \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.

```

3796     \bool_if:NT \l_@@_last_row_without_value_bool
3797     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3798     \tl_gput_right:Ne \g_@@_aux_tl
3799     {
3800       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3801       {
3802         \int_use:N \l_@@_first_row_int ,
3803         \int_use:N \c@iRow ,
3804         \int_use:N \g_@@_row_total_int ,
3805         \int_use:N \l_@@_first_col_int ,
3806         \int_use:N \c@jCol ,
3807         \int_use:N \g_@@_col_total_int
3808       }
3809     }

```

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`).

```

3810     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3811     {
3812       \tl_gput_right:Ne \g_@@_aux_tl
3813       {
3814         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3815         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3816       }
3817     }
3818     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3819     {
3820       \tl_gput_right:Ne \g_@@_aux_tl
3821       {

```

```

3822     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3823     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3824     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3825     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3826   }
3827 }

```

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

```

3828   \@@_create_diag_nodes:

```

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

```

3829   \pgfpicture
3830   \int_step_inline:nn \c@iRow
3831   {
3832     \pgfnodealias
3833     { \@@_env: - ##1 - last }
3834     { \@@_env: - ##1 - \int_use:N \c@jCol }
3835   }
3836   \int_step_inline:nn \c@jCol
3837   {
3838     \pgfnodealias
3839     { \@@_env: - last - ##1 }
3840     { \@@_env: - \int_use:N \c@iRow - ##1 }
3841   }
3842   \str_if_empty:NF \l_@@_name_str
3843   {
3844     \int_step_inline:nn \c@iRow
3845     {
3846       \pgfnodealias
3847       { \l_@@_name_str - ##1 - last }
3848       { \@@_env: - ##1 - \int_use:N \c@jCol }
3849     }
3850     \int_step_inline:nn \c@jCol
3851     {
3852       \pgfnodealias
3853       { \l_@@_name_str - last - ##1 }
3854       { \@@_env: - \int_use:N \c@iRow - ##1 }
3855     }
3856   }
3857   \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.

```

3858   \bool_if:NT \l_@@_parallelize_diags_bool
3859   {
3860     \int_gzero_new:N \g_@@_ddots_int
3861     \int_gzero_new:N \g_@@_iddots_int

```

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

```

3862     \dim_gzero_new:N \g_@@_delta_x_one_dim
3863     \dim_gzero_new:N \g_@@_delta_y_one_dim
3864     \dim_gzero_new:N \g_@@_delta_x_two_dim
3865     \dim_gzero_new:N \g_@@_delta_y_two_dim
3866   }
3867   \int_zero_new:N \l_@@_initial_i_int
3868   \int_zero_new:N \l_@@_initial_j_int

```

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

```

3869 \int_zero_new:N \l_@@_final_i_int
3870 \int_zero_new:N \l_@@_final_j_int
3871 \bool_set_false:N \l_@@_initial_open_bool
3872 \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.

```

3873 \bool_if:NT \l_@@_small_bool
3874 {
3875   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3876   \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.

```

3877   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3878     { 0.6 \l_@@_xdots_shorten_start_dim }
3879   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3880     { 0.6 \l_@@_xdots_shorten_end_dim }
3881 }

```

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

```

3882 \@@_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_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3883 \clist_if_empty:NF \l_@@_corners_clist \@@_compute_corners:

```

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

```

3884 \@@_adjust_pos_of_blocks_seq:
3885 \@@_deal_with_rounded_corners:
3886 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3887 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

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

```


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

```
3910 \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

We clear the list of the names of the potential `\SubMatrix` that will appear in the `\CodeAfter` (unfortunately, that list has to be global).

```
3911 \seq_gclear:N \g_@@_submatrix_names_seq
```

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

```
3912 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3913 { \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl }
```

And here's the `\CodeAfter`. Since the `\CodeAfter` may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command `\@@_CodeAfter_keys:`.

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

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

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

3938 \str_gclear:N \g_@@_name_env_str
3939 \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the 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`.

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

¹²e.g. `\color[rgb]{0.5,0.5,0}`

The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```

3942 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3943 { \keys_set:mn { nicematrix / CodeAfter } { #1 } }

```

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

```

3944 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3945 {
3946   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3947   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3948 }

```

The following command must *not* be protected.

```

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

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible”. That’s why we have to define the adequate version of `\@@_draw_dotted_lines`: whether Tikz is loaded or not (in that case, only PGF is loaded).

```

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

```

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

```

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

```

```

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

```

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

```

3990 \pgfdeclareshape { @@_diag_node }
3991 {
3992   \savedanchor { \five }
3993   {
3994     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3995     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3996   }
3997   \anchor { 5 } { \five }
3998   \anchor { center } { \pgfpointorigin }
3999   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4000   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4001   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4002   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4003   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4004   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4005   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4006   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4007   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4008   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4009 }

```

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).

```

4010 \cs_new_protected:Npn \@@_create_diag_nodes:
4011 {
4012   \pgfpicture
4013   \pgfrememberpicturepositiononpagetrue
4014   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4015   {
4016     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4017     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4018     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4019     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4020     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4021     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4022     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4023     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4024     \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.

```

4025     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4026     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4027     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4028     \str_if_empty:NF \l_@@_name_str
4029     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4030   }

```

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

```

4031     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4032     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4033     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4034     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4035     \pgfcoordinate

```

```

4036     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4037 \pgfnodealias
4038   { \@@_env: - last }
4039   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4040 \str_if_empty:NF \l_@@_name_str
4041   {
4042     \pgfnodealias
4043       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4044       { \@@_env: - \int_use:N \l_tmpa_int }
4045     \pgfnodealias
4046       { \l_@@_name_str - last }
4047       { \@@_env: - last }
4048   }
4049 \endpgfpicture
4050 }

```

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.

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

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

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

This command computes:

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

```

4051 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4052   {

```

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

```

4053   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4054   \int_set:Nn \l_@@_initial_i_int { #1 }
4055   \int_set:Nn \l_@@_initial_j_int { #2 }
4056   \int_set:Nn \l_@@_final_i_int { #1 }
4057   \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.

```

4058     \bool_set_false:N \l_@@_stop_loop_bool
4059     \bool_do_until:Nn \l_@@_stop_loop_bool
4060     {
4061         \int_add:Nn \l_@@_final_i_int { #3 }
4062         \int_add:Nn \l_@@_final_j_int { #4 }
4063         \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.

```

4064         \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4065         \if_int_compare:w #3 = \c_one_int
4066             \bool_set_true:N \l_@@_final_open_bool
4067         \else:
4068             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4069                 \bool_set_true:N \l_@@_final_open_bool
4070             \fi:
4071         \fi:
4072     \else:
4073         \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4074             \if_int_compare:w #4 = -1
4075                 \bool_set_true:N \l_@@_final_open_bool
4076             \fi:
4077         \else:
4078             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4079                 \if_int_compare:w #4 = \c_one_int
4080                     \bool_set_true:N \l_@@_final_open_bool
4081                 \fi:
4082             \fi:
4083         \fi:
4084     \fi:
4085     \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.

```

4086     {

```

We do a step backwards.

```

4087         \int_sub:Nn \l_@@_final_i_int { #3 }
4088         \int_sub:Nn \l_@@_final_j_int { #4 }
4089         \bool_set_true:N \l_@@_stop_loop_bool
4090     }

```

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`.

```

4091     {
4092         \cs_if_exist:cTF
4093         {
4094             @@ _ dotted _
4095             \int_use:N \l_@@_final_i_int -
4096             \int_use:N \l_@@_final_j_int
4097         }
4098         {
4099             \int_sub:Nn \l_@@_final_i_int { #3 }
4100             \int_sub:Nn \l_@@_final_j_int { #4 }
4101             \bool_set_true:N \l_@@_final_open_bool
4102             \bool_set_true:N \l_@@_stop_loop_bool
4103         }
4104         {
4105             \cs_if_exist:cTF
4106             {
4107                 pgf @ sh @ ns @ \@@_env:
4108                 - \int_use:N \l_@@_final_i_int

```

```

4109         - \int_use:N \l_@@_final_j_int
4110     }
4111     { \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.

```

4112     {
4113         \cs_set_nopar:cpn
4114         {
4115             @@ _ dotted _
4116             \int_use:N \l_@@_final_i_int -
4117             \int_use:N \l_@@_final_j_int
4118         }
4119         { }
4120     }
4121 }
4122 }
4123 }

```

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

```

4124 \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4125 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4126 \bool_do_until:Nn \l_@@_stop_loop_bool
4127 {
4128     \int_sub:Nn \l_@@_initial_i_int { #3 }
4129     \int_sub:Nn \l_@@_initial_j_int { #4 }
4130     \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.

```

4131 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4132 \if_int_compare:w #3 = \c_one_int
4133 \bool_set_true:N \l_@@_initial_open_bool
4134 \else:

```

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

```

4135 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4136 \bool_set_true:N \l_@@_initial_open_bool
4137 \fi:
4138 \fi:
4139 \else:
4140 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4141 \if_int_compare:w #4 = \c_one_int
4142 \bool_set_true:N \l_@@_initial_open_bool
4143 \fi:
4144 \else:
4145 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4146 \if_int_compare:w #4 = -1
4147 \bool_set_true:N \l_@@_initial_open_bool
4148 \fi:
4149 \fi:
4150 \fi:
4151 \fi:

```

```

4152 \bool_if:NTF \l_@@_initial_open_bool
4153 {
4154   \int_add:Nn \l_@@_initial_i_int { #3 }
4155   \int_add:Nn \l_@@_initial_j_int { #4 }
4156   \bool_set_true:N \l_@@_stop_loop_bool
4157 }
4158 {
4159   \cs_if_exist:cTF
4160   {
4161     @@ _ dotted _
4162     \int_use:N \l_@@_initial_i_int -
4163     \int_use:N \l_@@_initial_j_int
4164   }
4165   {
4166     \int_add:Nn \l_@@_initial_i_int { #3 }
4167     \int_add:Nn \l_@@_initial_j_int { #4 }
4168     \bool_set_true:N \l_@@_initial_open_bool
4169     \bool_set_true:N \l_@@_stop_loop_bool
4170   }
4171   {
4172     \cs_if_exist:cTF
4173     {
4174       pgf @ sh @ ns @ \@@_env:
4175       - \int_use:N \l_@@_initial_i_int
4176       - \int_use:N \l_@@_initial_j_int
4177     }
4178     { \bool_set_true:N \l_@@_stop_loop_bool }
4179     {
4180       \cs_set_nopar:cpn
4181       {
4182         @@ _ dotted _
4183         \int_use:N \l_@@_initial_i_int -
4184         \int_use:N \l_@@_initial_j_int
4185       }
4186       { }
4187     }
4188   }
4189 }
4190 }

```

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.

```

4191 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4192 {
4193   { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4194   { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4195   { \int_use:N \l_@@_final_i_int }
4196   { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4197   { } % for the name of the block
4198 }
4199 }

```

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.

```

4200 \cs_new_protected:Npn \@@_open_shorten:
4201 {

```

```

4202 \bool_if:NT \l_@@_initial_open_bool
4203   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4204 \bool_if:NT \l_@@_final_open_bool
4205   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4206 }

```

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).

```

4207 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4208   {
4209     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4210     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4211     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4212     \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`.

```

4213 \seq_if_empty:NF \g_@@_submatrix_seq
4214   {
4215     \seq_map_inline:Nn \g_@@_submatrix_seq
4216       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4217   }
4218 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix we are analyzing.

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

```

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

```

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

```

4219 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4220   {
4221     \if_int_compare:w #3 > #1
4222     \else:
4223       \if_int_compare:w #1 > #5
4224       \else:
4225         \if_int_compare:w #4 > #2
4226         \else:
4227           \if_int_compare:w #2 > #6
4228           \else:
4229             \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4230             \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4231             \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4232             \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:

```



```

4233     \fi:
4234     \fi:
4235     \fi:
4236     \fi:
4237 }

4238 \cs_new_protected:Npn \@@_set_initial_coords:
4239 {
4240     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4241     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4242 }
4243 \cs_new_protected:Npn \@@_set_final_coords:
4244 {
4245     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4246     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4247 }
4248 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4249 {
4250     \pgfpointanchor
4251     {
4252         \@@_env:
4253         - \int_use:N \l_@@_initial_i_int
4254         - \int_use:N \l_@@_initial_j_int
4255     }
4256     { #1 }
4257     \@@_set_initial_coords:
4258 }
4259 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4260 {
4261     \pgfpointanchor
4262     {
4263         \@@_env:
4264         - \int_use:N \l_@@_final_i_int
4265         - \int_use:N \l_@@_final_j_int
4266     }
4267     { #1 }
4268     \@@_set_final_coords:
4269 }

4270 \cs_new_protected:Npn \@@_open_x_initial_dim:
4271 {
4272     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4273     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4274     {
4275         \cs_if_exist:cT
4276         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4277         {
4278             \pgfpointanchor
4279             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4280             { west }
4281             \dim_set:Nn \l_@@_x_initial_dim
4282             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4283         }
4284     }
}

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

4285     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4286     {
4287         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4288         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4289         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4290     }
4291 }

```

```

4292 \cs_new_protected:Npn \@@_open_x_final_dim:
4293 {
4294   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4295   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4296   {
4297     \cs_if_exist:cT
4298     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4299     {
4300       \pgfpointanchor
4301       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4302       { east }
4303       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4304       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4305     }
4306   }

```

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

```

4307   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4308   {
4309     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4310     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4311     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4312   }
4313 }

```

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.

```

4314 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4315 {
4316   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4317   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4318   {
4319     \@@_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.

```

4320   \group_begin:
4321   \@@_open_shorten:
4322   \int_if_zero:nTF { #1 }
4323   { \color { nicematrix-first-row } }
4324   {

```

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.

```

4325     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4326     { \color { nicematrix-last-row } }
4327   }
4328   \keys_set:nn { nicematrix / xdots } { #3 }
4329   \@@_color:o \l_@@_xdots_color_tl
4330   \@@_actually_draw_Ldots:
4331   \group_end:
4332 }
4333 }

```

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`.

```

4334 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4335   {
4336     \bool_if:NTF \l_@@_initial_open_bool
4337     {
4338       \@@_open_x_initial_dim:
4339       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4340       \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4341     }
4342     { \@@_set_initial_coords_from_anchor:n { base-east } }
4343     \bool_if:NTF \l_@@_final_open_bool
4344     {
4345       \@@_open_x_final_dim:
4346       \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4347       \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4348     }
4349     { \@@_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.

```

4350     \bool_lazy_all:nTF
4351     {
4352       \l_@@_initial_open_bool
4353       \l_@@_final_open_bool
4354       { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4355     }
4356     {
4357       \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4358       \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4359     }

```

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

```

4360     {
4361       \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4362       \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4363     }
4364     \@@_draw_line:
4365   }

```

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.

```

4366 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4367   {
4368     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4369     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4370     {
4371       \@@_find_extremities_of_line:nmmn { #1 } { #2 } 0 1

```

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

```

4372     \group_begin:
4373     \@@_open_shorten:
4374     \int_if_zero:nTF { #1 }
4375     { \color { nicematrix-first-row } }
4376     {

```

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.

```

4377         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4378         { \color { nicematrix-last-row } }
4379     }
4380     \keys_set:nn { nicematrix / xdots } { #3 }
4381     \@@_color:o \l_@@_xdots_color_tl
4382     \@@_actually_draw_Cdots:
4383     \group_end:
4384 }
4385 }

```

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`.

```

4386 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4387 {
4388     \bool_if:NTF \l_@@_initial_open_bool
4389     { \@@_open_x_initial_dim: }
4390     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4391     \bool_if:NTF \l_@@_final_open_bool
4392     { \@@_open_x_final_dim: }
4393     { \@@_set_final_coords_from_anchor:n { mid-west } }
4394     \bool_lazy_and:nnTF
4395     \l_@@_initial_open_bool
4396     \l_@@_final_open_bool
4397     {
4398         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4399         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4400         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4401         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4402         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4403     }
4404     {
4405         \bool_if:NT \l_@@_initial_open_bool
4406         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4407         \bool_if:NT \l_@@_final_open_bool
4408         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4409     }
4410     \@@_draw_line:
4411 }
4412 \cs_new_protected:Npn \@@_open_y_initial_dim:
4413 {
4414     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4415     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4416     {
4417         \cs_if_exist:cT
4418         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4419         {
4420             \pgfpointanchor
4421             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4422             { north }
4423             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim

```

```

4424         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4425     }
4426 }
4427 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4428 {
4429     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4430     \dim_set:Nn \l_@@_y_initial_dim
4431     {
4432         \fp_to_dim:n
4433         {
4434             \pgf@y
4435             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4436         }
4437     }
4438 }
4439 }
4440 \cs_new_protected:Npn \@@_open_y_final_dim:
4441 {
4442     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4443     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4444     {
4445         \cs_if_exist:cT
4446         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4447         {
4448             \pgfpointanchor
4449             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4450             { south }
4451             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4452             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4453         }
4454     }
4455     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4456     {
4457         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4458         \dim_set:Nn \l_@@_y_final_dim
4459         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4460     }
4461 }

```

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.

```

4462 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4463 {
4464     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4465     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4466     {
4467         \@@_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.

```

4468     \group_begin:
4469     \@@_open_shorten:
4470     \int_if_zero:nTF { #2 }
4471     { \color { nicematrix-first-col } }
4472     {
4473         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4474         { \color { nicematrix-last-col } }
4475     }
4476     \keys_set:nn { nicematrix / xdots } { #3 }
4477     \@@_color:o \l_@@_xdots_color_tl
4478     \@@_actually_draw_Vdots:
4479     \group_end:
4480 }
4481 }

```

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`.

```
4482 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4483 {
```

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

```
4484   \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.

```
4485   {
4486     \@@_open_y_initial_dim:
4487     \@@_open_y_final_dim:
4488     \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4489     {
4490       \@@_qpoint:n { col - 1 }
4491       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4492       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4493       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4494       \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4495     }
4496     {
4497       \bool_lazy_and:nnTF
4498       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4499       { \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”.

```
4500     {
4501       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4502       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4503       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4504       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4505       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4506     }
```

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

```
4507     {
4508       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4509       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4510       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4511       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4512     }
4513   }
4514 }
```

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

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```
4515   {
4516     \bool_set_false:N \l_tmpa_bool
4517     \bool_if:NF \l_@@_initial_open_bool
4518     {
4519       \bool_if:NF \l_@@_final_open_bool
4520       {
```

```

4521         \@@_set_initial_coords_from_anchor:n { south-west }
4522         \@@_set_final_coords_from_anchor:n { north-west }
4523         \bool_set:Nn \l_tmpa_bool
4524         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4525     }
4526 }

```

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

```

4527     \bool_if:NTF \l_@@_initial_open_bool
4528     {
4529         \@@_open_y_initial_dim:
4530         \@@_set_final_coords_from_anchor:n { north }
4531         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4532     }
4533     {
4534         \@@_set_initial_coords_from_anchor:n { south }
4535         \bool_if:NTF \l_@@_final_open_bool
4536         \@@_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.

```

4537     {
4538         \@@_set_final_coords_from_anchor:n { north }
4539         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4540         {
4541             \dim_set:Nn \l_@@_x_initial_dim
4542             {
4543                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4544                 \l_@@_x_initial_dim \l_@@_x_final_dim
4545             }
4546         }
4547     }
4548 }
4549 }
4550 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4551 \@@_draw_line:
4552 }

```

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.

```

4553 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4554 {
4555     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4556     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4557     {
4558         \@@_find_extremities_of_line:nmmn { #1 } { #2 } 1 1

```

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

```

4559     \group_begin:
4560     \@@_open_shorten:
4561     \keys_set:nn { nicematrix / xdots } { #3 }
4562     \@@_color:o \l_@@_xdots_color_tl
4563     \@@_actually_draw_Ddots:
4564     \group_end:
4565 }
4566 }

```

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`.

```

4567 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4568 {
4569   \bool_if:NTF \l_@@_initial_open_bool
4570   {
4571     \@@_open_y_initial_dim:
4572     \@@_open_x_initial_dim:
4573   }
4574   { \@@_set_initial_coords_from_anchor:n { south~east } }
4575   \bool_if:NTF \l_@@_final_open_bool
4576   {
4577     \@@_open_x_final_dim:
4578     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4579   }
4580   { \@@_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.

```

4581   \bool_if:NT \l_@@_parallelize_diags_bool
4582   {
4583     \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).

```

4584     \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.

```

4585     {
4586       \dim_gset:Nn \g_@@_delta_x_one_dim
4587       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4588       \dim_gset:Nn \g_@@_delta_y_one_dim
4589       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4590     }

```

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`.

```

4591     {
4592       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4593       {
4594         \dim_set:Nn \l_@@_y_final_dim
4595         {
4596           \l_@@_y_initial_dim +
4597           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4598           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4599         }
4600       }
4601     }
4602   }
4603   \@@_draw_line:
4604 }

```


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.

```

4605 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4606 {
4607   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4608   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4609   {
4610     \@@_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.

```

4611   \group_begin:
4612   \@@_open_shorten:
4613   \keys_set:nn { nicematrix / xdots } { #3 }
4614   \@@_color:o \l_@@_xdots_color_tl
4615   \@@_actually_draw_Iddots:
4616   \group_end:
4617 }
4618 }

```

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`.

```

4619 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4620 {
4621   \bool_if:NTF \l_@@_initial_open_bool
4622   {
4623     \@@_open_y_initial_dim:
4624     \@@_open_x_initial_dim:
4625   }
4626   { \@@_set_initial_coords_from_anchor:n { south~west } }
4627   \bool_if:NTF \l_@@_final_open_bool
4628   {
4629     \@@_open_y_final_dim:
4630     \@@_open_x_final_dim:
4631   }
4632   { \@@_set_final_coords_from_anchor:n { north~east } }
4633   \bool_if:NT \l_@@_parallelize_diags_bool
4634   {
4635     \int_gincr:N \g_@@_iddots_int
4636     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4637     {
4638       \dim_gset:Nn \g_@@_delta_x_two_dim
4639       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4640       \dim_gset:Nn \g_@@_delta_y_two_dim
4641       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4642     }
4643     {
4644       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4645       {
4646         \dim_set:Nn \l_@@_y_final_dim
4647         {
4648           \l_@@_y_initial_dim +

```

```

4649         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4650         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4651     }
4652 }
4653 }
4654 }
4655 \@@_draw_line:
4656 }

```

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`

```

4657 \cs_new_protected:Npn \@@_draw_line:
4658 {
4659     \pgfrememberpicturepositiononpagetrue
4660     \pgf@relevantforpicturesizefalse
4661     \bool_lazy_or:nnTF
4662     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4663     \l_@@_dotted_bool
4664     \@@_draw_standard_dotted_line:
4665     \@@_draw_unstandard_dotted_line:
4666 }

```

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.

```

4667 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4668 {
4669     \begin { scope }
4670     \@@_draw_unstandard_dotted_line:o
4671     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4672 }

```

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.

```

4673 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4674 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4675 {
4676     \@@_draw_unstandard_dotted_line:nooo
4677     { #1 }
4678     \l_@@_xdots_up_tl
4679     \l_@@_xdots_down_tl
4680     \l_@@_xdots_middle_tl
4681 }

```

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

```

4682 \hook_gput_code:nnn { begindocument } { . }
4683 {
4684   \IfPackageLoadedT { tikz }
4685   {
4686     \tikzset
4687     {
4688       @@_node_above / .style = { sloped , above } ,
4689       @@_node_below / .style = { sloped , below } ,
4690       @@_node_middle / .style =
4691       {
4692         sloped ,
4693         inner-sep = \c_@@_innersep_middle_dim
4694       }
4695     }
4696   }
4697 }

4698 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4699 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4700 {

```

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.

```

4701   \dim_zero_new:N \l_@@_l_dim
4702   \dim_set:Nn \l_@@_l_dim
4703   {
4704     \fp_to_dim:n
4705     {
4706       sqrt
4707       (
4708         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4709         +
4710         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4711       )
4712     }
4713   }

```

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.

```

4714   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4715   {
4716     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4717     \@@_draw_unstandard_dotted_line_i:
4718   }

```

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

```

4719   \bool_if:NT \l_@@_xdots_h_labels_bool
4720   {
4721     \tikzset
4722     {
4723       @@_node_above / .style = { auto = left } ,
4724       @@_node_below / .style = { auto = right } ,
4725       @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4726     }
4727   }

```

```

4728 \tl_if_empty:nF { #4 }
4729   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4730 \draw
4731   [ #1 ]
4732   ( \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).

```

4733   -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4734   node [ @@_node_below ] { $ \scriptstyle #3 $ }
4735   node [ @@_node_above ] { $ \scriptstyle #2 $ }
4736   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4737 \end { scope }
4738 }

4739 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4740 {
4741   \dim_set:Nn \l_tmpa_dim
4742   {
4743     \l_@@_x_initial_dim
4744     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4745     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4746   }
4747   \dim_set:Nn \l_tmpb_dim
4748   {
4749     \l_@@_y_initial_dim
4750     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4751     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4752   }
4753   \dim_set:Nn \l_@@_tmpc_dim
4754   {
4755     \l_@@_x_final_dim
4756     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4757     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4758   }
4759   \dim_set:Nn \l_@@_tmpd_dim
4760   {
4761     \l_@@_y_final_dim
4762     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4763     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4764   }
4765   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4766   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4767   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4768   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4769 }

```

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

```

4770 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4771 {
4772   \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.

```

4773   \dim_zero_new:N \l_@@_l_dim
4774   \dim_set:Nn \l_@@_l_dim
4775   {
4776     \fp_to_dim:n
4777     {
4778       sqrt
4779       (
4780         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4781         +

```

```

4782         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4783     )
4784 }
4785 }

```

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.

```

4786 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4787 {
4788     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4789     \@@_draw_standard_dotted_line_i:
4790 }
4791 \group_end:
4792 \bool_lazy_all:nF
4793 {
4794     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4795     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4796     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4797 }
4798 \l_@@_labels_standard_dotted_line:
4799 }
4800 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4801 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4802 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4803 \int_set:Nn \l_tmpa_int
4804 {
4805     \dim_ratio:nn
4806     {
4807         \l_@@_l_dim
4808         - \l_@@_xdots_shorten_start_dim
4809         - \l_@@_xdots_shorten_end_dim
4810     }
4811     \l_@@_xdots_inter_dim
4812 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4813 \dim_set:Nn \l_tmpa_dim
4814 {
4815     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4816     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4817 }
4818 \dim_set:Nn \l_tmpb_dim
4819 {
4820     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4821     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4822 }

```

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.

```

4823 \dim_gadd:Nn \l_@@_x_initial_dim
4824 {
4825     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4826     \dim_ratio:nn
4827     {
4828         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4829         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4830     }

```

```

4831     { 2 \l_@@_l_dim }
4832   }
4833   \dim_gadd:Nn \l_@@_y_initial_dim
4834   {
4835     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4836     \dim_ratio:nn
4837     {
4838       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4839       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4840     }
4841     { 2 \l_@@_l_dim }
4842   }
4843   \pgf@relevantforpicturesizefalse
4844   \int_step_inline:nnn \c_zero_int \l_tmpa_int
4845   {
4846     \pgfpathcircle
4847     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4848     { \l_@@_xdots_radius_dim }
4849     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4850     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4851   }
4852   \pgfusepathqfill
4853 }

```

```

4854 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4855 {
4856   \pgfscope
4857   \pgftransformshift
4858   {
4859     \pgfpointlineattime { 0.5 }
4860     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4861     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4862   }
4863   \fp_set:Nn \l_tmpa_fp
4864   {
4865     atand
4866     (
4867       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4868       \l_@@_x_final_dim - \l_@@_x_initial_dim
4869     )
4870   }
4871   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4872   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4873   \tl_if_empty:NF \l_@@_xdots_middle_tl
4874   {
4875     \begin { pgfscope }
4876     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4877     \pgfnode
4878     { rectangle }
4879     { center }
4880     {
4881       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4882       {
4883         \c_math_toggle_token
4884         \scriptstyle \l_@@_xdots_middle_tl
4885         \c_math_toggle_token
4886       }
4887     }
4888     { }
4889     {
4890       \pgfsetfillcolor { white }
4891       \pgfusepath { fill }
4892     }

```

```

4893     \end { pgfscope }
4894   }
4895   \tl_if_empty:NF \l_@@_xdots_up_tl
4896   {
4897     \pgfnode
4898     { rectangle }
4899     { south }
4900     {
4901       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4902       {
4903         \c_math_toggle_token
4904         \scriptstyle \l_@@_xdots_up_tl
4905         \c_math_toggle_token
4906       }
4907     }
4908     { }
4909     { \pgfusepath { } }
4910   }
4911   \tl_if_empty:NF \l_@@_xdots_down_tl
4912   {
4913     \pgfnode
4914     { rectangle }
4915     { north }
4916     {
4917       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4918       {
4919         \c_math_toggle_token
4920         \scriptstyle \l_@@_xdots_down_tl
4921         \c_math_toggle_token
4922       }
4923     }
4924     { }
4925     { \pgfusepath { } }
4926   }
4927   \endpgfscope
4928 }

```

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.

```

4929 \hook_gput_code:nnn { begindocument } { . }
4930 {
4931   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4932   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4933   \cs_new_protected:Npn \@@_Ldots
4934   { \@@_collect_options:n { \@@_Ldots_i } }
4935   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4936   {
4937     \int_if_zero:nTF \c@jCol
4938     { \@@_error:nn { in-first-col } \Ldots }
4939     {

```

```

4940     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4941     { \@@_error:nn { in~last~col } \Ldots }
4942     {
4943       \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4944       { #1 , down = #2 , up = #3 , middle = #4 }
4945     }
4946   }
4947   \bool_if:NF \l_@@_nullify_dots_bool
4948   { \phantom { \ensuremath { \@@_old_ldots } } }
4949   \bool_gset_true:N \g_@@_empty_cell_bool
4950 }

4951 \cs_new_protected:Npn \@@_Cdots
4952 { \@@_collect_options:n { \@@_Cdots_i } }
4953 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4954 {
4955   \int_if_zero:nTF \c@jCol
4956   { \@@_error:nn { in~first~col } \Cdots }
4957   {
4958     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4959     { \@@_error:nn { in~last~col } \Cdots }
4960     {
4961       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4962       { #1 , down = #2 , up = #3 , middle = #4 }
4963     }
4964   }
4965   \bool_if:NF \l_@@_nullify_dots_bool
4966   { \phantom { \ensuremath { \@@_old_cdots } } }
4967   \bool_gset_true:N \g_@@_empty_cell_bool
4968 }

4969 \cs_new_protected:Npn \@@_Vdots
4970 { \@@_collect_options:n { \@@_Vdots_i } }
4971 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4972 {
4973   \int_if_zero:nTF \c@iRow
4974   { \@@_error:nn { in~first~row } \Vdots }
4975   {
4976     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4977     { \@@_error:nn { in~last~row } \Vdots }
4978     {
4979       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4980       { #1 , down = #2 , up = #3 , middle = #4 }
4981     }
4982   }
4983   \bool_if:NF \l_@@_nullify_dots_bool
4984   { \phantom { \ensuremath { \@@_old_vdots } } }
4985   \bool_gset_true:N \g_@@_empty_cell_bool
4986 }

4987 \cs_new_protected:Npn \@@_Ddots
4988 { \@@_collect_options:n { \@@_Ddots_i } }
4989 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4990 {
4991   \int_case:nnF \c@iRow
4992   {
4993     0 { \@@_error:nn { in~first~row } \Ddots }
4994     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4995   }
4996   {
4997     \int_case:nnF \c@jCol

```



```

4998     {
4999         0           { \@@_error:nn { in~first~col } \Ddots }
5000         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5001     }
5002     {
5003         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5004         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5005         { #1 , down = #2 , up = #3 , middle = #4 }
5006     }
5007
5008 }
5009 \bool_if:NF \l_@@_nullify_dots_bool
5010 { \phantom { \ensuremath { \@@_old_ddots } } }
5011 \bool_gset_true:N \g_@@_empty_cell_bool
5012 }

5013 \cs_new_protected:Npn \@@_Iddots
5014 { \@@_collect_options:n { \@@_Iddots_i } }
5015 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5016 {
5017     \int_case:nnF \c@iRow
5018     {
5019         0           { \@@_error:nn { in~first~row } \Iddots }
5020         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5021     }
5022     {
5023         \int_case:nnF \c@jCol
5024         {
5025             0           { \@@_error:nn { in~first~col } \Iddots }
5026             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5027         }
5028         {
5029             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5030             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5031             { #1 , down = #2 , up = #3 , middle = #4 }
5032         }
5033     }
5034     \bool_if:NF \l_@@_nullify_dots_bool
5035     { \phantom { \ensuremath { \@@_old_iddots } } }
5036     \bool_gset_true:N \g_@@_empty_cell_bool
5037 }
5038 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5039 \keys_define:nn { nicematrix / Ddots }
5040 {
5041     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5042     draw-first .default:n = true ,
5043     draw-first .value_forbidden:n = true
5044 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5045 \cs_new_protected:Npn \@@_Hspace:
5046 {
5047     \bool_gset_true:N \g_@@_empty_cell_bool
5048     \hspace
5049 }

```

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.

```

5050 \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`.

```

5051 \cs_new:Npn \@@_Hdotsfor:
5052 {
5053   \bool_lazy_and:nnTF
5054     { \int_if_zero_p:n \c@jCol }
5055     { \int_if_zero_p:n \l_@@_first_col_int }
5056     {
5057       \bool_if:NTF \g_@@_after_col_zero_bool
5058         {
5059           \multicolumn { 1 } { c } { }
5060           \@@_Hdotsfor_i
5061         }
5062         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5063       }
5064     {
5065       \multicolumn { 1 } { c } { }
5066       \@@_Hdotsfor_i
5067     }
5068 }

```

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:`).

```

5069 \hook_gput_code:nnn { begindocument } { . }
5070 {
5071   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5072   \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.

```

5073   \cs_new_protected:Npn \@@_Hdotsfor_i
5074     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5075   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5076     {
5077     \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5078       {
5079         \@@_Hdotsfor:nnnn
5080         { \int_use:N \c@iRow }
5081         { \int_use:N \c@jCol }
5082         { #2 }
5083         {
5084           #1 , #3 ,
5085           down = \exp_not:n { #4 } ,
5086           up = \exp_not:n { #5 } ,
5087           middle = \exp_not:n { #6 }
5088         }
5089       }
5090     \prg_replicate:nn { #2 - 1 }
5091     {
5092       &
5093       \multicolumn { 1 } { c } { }
5094       \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5095     }
5096   }
5097 }

5098 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5099 {
5100   \bool_set_false:N \l_@@_initial_open_bool
5101   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```
5102 \int_set:Nn \l_@@_initial_i_int { #1 }
5103 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
5104 \int_compare:nNnTF { #2 } = \c_one_int
5105 {
5106   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5107   \bool_set_true:N \l_@@_initial_open_bool
5108 }
5109 {
5110   \cs_if_exist:cTF
5111   {
5112     pgf @ sh @ ns @ \@@_env:
5113     - \int_use:N \l_@@_initial_i_int
5114     - \int_eval:n { #2 - 1 }
5115   }
5116   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5117   {
5118     \int_set:Nn \l_@@_initial_j_int { #2 }
5119     \bool_set_true:N \l_@@_initial_open_bool
5120   }
5121 }
5122 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5123 {
5124   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5125   \bool_set_true:N \l_@@_final_open_bool
5126 }
5127 {
5128   \cs_if_exist:cTF
5129   {
5130     pgf @ sh @ ns @ \@@_env:
5131     - \int_use:N \l_@@_final_i_int
5132     - \int_eval:n { #2 + #3 }
5133   }
5134   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5135   {
5136     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5137     \bool_set_true:N \l_@@_final_open_bool
5138   }
5139 }
5140 \group_begin:
5141 \@@_open_shorten:
5142 \int_if_zero:nTF { #1 }
5143 { \color { nicematrix-first-row } }
5144 {
5145   \int_compare:nNnT { #1 } = \g_@@_row_total_int
5146   { \color { nicematrix-last-row } }
5147 }
5148
5149 \keys_set:nn { nicematrix / xdots } { #4 }
5150 \@@_color:o \l_@@_xdots_color_tl
5151 \@@_actually_draw_ldots:
5152 \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).

```
5153 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5154 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5155 }
```

```

5156 \hook_gput_code:nnn { begindocument } { . }
5157 {
5158   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5159   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5160   \cs_new_protected:Npn \@@_Vdotsfor:
5161     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5162   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5163     {
5164     \bool_gset_true:N \g_@@_empty_cell_bool
5165     \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5166       {
5167         \@@_Vdotsfor:nnnn
5168         { \int_use:N \c@iRow }
5169         { \int_use:N \c@jCol }
5170         { #2 }
5171         {
5172           #1 , #3 ,
5173           down = \exp_not:n { #4 } ,
5174           up = \exp_not:n { #5 } ,
5175           middle = \exp_not:n { #6 }
5176         }
5177       }
5178     }
5179 }

5180 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5181 {
5182   \bool_set_false:N \l_@@_initial_open_bool
5183   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5184   \int_set:Nn \l_@@_initial_j_int { #2 }
5185   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5186   \int_compare:nNnTF { #1 } = \c_one_int
5187   {
5188     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5189     \bool_set_true:N \l_@@_initial_open_bool
5190   }
5191   {
5192     \cs_if_exist:cTF
5193     {
5194       pgf @ sh @ ns @ \@@_env:
5195       - \int_eval:n { #1 - 1 }
5196       - \int_use:N \l_@@_initial_j_int
5197     }
5198     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5199     {
5200       \int_set:Nn \l_@@_initial_i_int { #1 }
5201       \bool_set_true:N \l_@@_initial_open_bool
5202     }
5203   }
5204   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5205   {
5206     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5207     \bool_set_true:N \l_@@_final_open_bool
5208   }
5209   {
5210     \cs_if_exist:cTF
5211     {
5212       pgf @ sh @ ns @ \@@_env:
5213       - \int_eval:n { #1 + #3 }

```

```

5214         - \int_use:N \l_@@_final_j_int
5215     }
5216     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5217     {
5218         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5219         \bool_set_true:N \l_@@_final_open_bool
5220     }
5221 }
5222 \group_begin:
5223 \@@_open_shorten:
5224 \int_if_zero:nTF { #2 }
5225   { \color { nicematrix-first-col } }
5226   {
5227     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5228     { \color { nicematrix-last-col } }
5229   }
5230 \keys_set:nn { nicematrix / xdots } { #4 }
5231 \@@_color:o \l_@@_xdots_color_tl
5232 \@@_actually_draw_Vdots:
5233 \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).

```

5234 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5235   { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5236 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5237 \NewDocumentCommand \@@_rotate: { 0 { } }
5238 {
5239   \peek_remove_spaces:n
5240   {
5241     \bool_gset_true:N \g_@@_rotate_bool
5242     \keys_set:nn { nicematrix / rotate } { #1 }
5243   }
5244 }
5245 \keys_define:nn { nicematrix / rotate }
5246 {
5247   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5248   c .value_forbidden:n = true ,
5249   unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5250 }

```

19 The command `\line` accessible in code-after

In the `\CodeAfter`, the command `\@@_line:nn` will be linked to `\line`. This command takes two arguments which are the specifications of two cells in the array (in the format $i-j$) and draws a dotted line between these cells. In fact, it also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format $i-j$, our command applies the command `\int_eval:n` to i and j ;

- If not (that is to say, when it’s a name of a `\Block`), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).¹³

```

5251 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5252 {
5253   \tl_if_empty:nTF { #2 }
5254     { #1 }
5255     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5256 }
5257 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5258 { \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).

```

5259 \hook_gput_code:nnn { begindocument } { . }
5260 {
5261   \cs_set_nopar:Npn \l_@@_argspec_tl
5262     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5263   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5264   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5265     {
5266       \group_begin:
5267       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5268       \@@_color:o \l_@@_xdots_color_tl
5269       \use:e
5270       {
5271         \@@_line_i:nn
5272         { \@@_double_int_eval:n #2 - \q_stop }
5273         { \@@_double_int_eval:n #3 - \q_stop }
5274       }
5275       \group_end:
5276     }
5277 }
5278 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5279 {
5280   \bool_set_false:N \l_@@_initial_open_bool
5281   \bool_set_false:N \l_@@_final_open_bool
5282   \bool_lazy_or:nnTF
5283     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5284     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5285     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5286   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5287 }
5288 \hook_gput_code:nnn { begindocument } { . }
5289 {
5290   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5291   {

```

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:.`

```

5292     \c_@@_pgfortikzpicture_tl
5293     \@@_draw_line_iii:nn { #1 } { #2 }
5294     \c_@@_endpgfortikzpicture_tl
5295   }
5296 }

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

5297 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5298 {
5299   \pgfrememberpicturepositiononpagetrue
5300   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5301   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5302   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5303   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5304   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5305   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5306   \@@_draw_line:
5307 }

```

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`

```

5308 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5309 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5310 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5311 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5312 {
5313   \tl_gput_right:Ne \g_@@_row_style_tl
5314 }

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5315   \exp_not:N
5316   \@@_if_row_less_than:nn
5317   { \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`).

```

5318   { \exp_not:n { #1 } \scan_stop: }
5319 }
5320 }

```

```

5321 \keys_define:nn { nicematrix / RowStyle }
5322 {
5323   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5324   cell-space-top-limit .value_required:n = true ,
5325   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5326   cell-space-bottom-limit .value_required:n = true ,
5327   cell-space-limits .meta:n =
5328   {

```

```

5329     cell-space-top-limit = #1 ,
5330     cell-space-bottom-limit = #1 ,
5331   } ,
5332   color .tl_set:N = \l_@@_color_tl ,
5333   color .value_required:n = true ,
5334   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5335   bold .default:n = true ,
5336   nb-rows .code:n =
5337     \str_if_eq:eeTF { #1 } { * }
5338     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5339     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5340   nb-rows .value_required:n = true ,
5341   rowcolor .tl_set:N = \l_tmpa_tl ,
5342   rowcolor .value_required:n = true ,
5343   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5344 }

```

```

5345 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5346 {
5347   \group_begin:
5348   \tl_clear:N \l_tmpa_tl
5349   \tl_clear:N \l_@@_color_tl
5350   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5351   \dim_zero:N \l_tmpa_dim
5352   \dim_zero:N \l_tmpb_dim
5353   \keys_set:nm { nicematrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5354   \tl_if_empty:NF \l_tmpa_tl
5355   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5356     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5357     {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5358         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5359         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5360         { \int_use:N \c@iRow - * }
5361     }

```

Then, the other rows (if there is several rows).

```

5362     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5363     {
5364         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5365         {
5366             \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5367             {
5368                 \int_eval:n { \c@iRow + 1 }
5369                 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5370             }
5371         }
5372     }
5373 }
5374 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5375   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5376   {
5377     \@@_put_in_row_style:e
5378     {
5379       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5380       {

```


It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5381         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5382         { \dim_use:N \l_tmpa_dim }
5383     }
5384 }
5385 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5386     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5387     {
5388         \@@_put_in_row_style:e
5389         {
5390             \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5391             {
5392                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5393                 { \dim_use:N \l_tmpb_dim }
5394             }
5395         }
5396     }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5397     \tl_if_empty:NF \l_@@_color_tl
5398     {
5399         \@@_put_in_row_style:e
5400         {
5401             \mode_leave_vertical:
5402             \@@_color:n { \l_@@_color_tl }
5403         }
5404     }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5405     \bool_if:NT \l_@@_bold_row_style_bool
5406     {
5407         \@@_put_in_row_style:n
5408         {
5409             \exp_not:n
5410             {
5411                 \if_mode_math:
5412                 \c_math_toggle_token
5413                 \bfseries \boldmath
5414                 \c_math_toggle_token
5415                 \else:
5416                 \bfseries \boldmath
5417                 \fi:
5418             }
5419         }
5420     }
5421     \group_end:
5422     \g_@@_row_style_tl
5423     \ignorespaces
5424 }

```

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).

```
5425 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5426 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5427 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5428 {
```

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.

```
5429 \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`.

```
5430 \str_if_in:nnF { #1 } { !! }
5431 {
5432 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5433 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5434 }
5435 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5436 {
5437 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5438 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5439 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5440 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5441 }
```

The following command must be used within a `\pgfpicture`.

```
5442 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5443 {
5444 \dim_compare:nnNt \l_@@_tab_rounded_corners_dim > \c_zero_dim
5445 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5446 \group_begin:
5447 \pgfsetcornersarced
5448 {
5449 \pgfpoint
5450 { \l_@@_tab_rounded_corners_dim }
5451 { \l_@@_tab_rounded_corners_dim }
5452 }
```

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.

```

5453     \bool_if:NTF \l_@@_hvlines_bool
5454     {
5455         \pgfpathrectanglecorners
5456         {
5457             \pgfpointadd
5458             { \@@_qpoint:n { row-1 } }
5459             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5460         }
5461         {
5462             \pgfpointadd
5463             {
5464                 \@@_qpoint:n
5465                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5466             }
5467             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5468         }
5469     }
5470     {
5471         \pgfpathrectanglecorners
5472         { \@@_qpoint:n { row-1 } }
5473         {
5474             \pgfpointadd
5475             {
5476                 \@@_qpoint:n
5477                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5478             }
5479             { \pgfpoint \c_zero_dim \arrayrulewidth }
5480         }
5481     }
5482     \pgfusepath { clip }
5483     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5484     }
5485 }

```

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`).

```

5486 \cs_new_protected:Npn \@@_actually_color:
5487 {
5488     \pgfpicture
5489     \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.

```

5490     \@@_clip_with_rounded_corners:
5491     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5492     {
5493         \int_compare:nNnTF { ##1 } = \c_one_int
5494         {
5495             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5496             \use:c { g_@@_color _ 1 _tl }
5497             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5498         }
5499         {
5500             \begin { pgfscope }
5501                 \@@_color_opacity ##2
5502                 \use:c { g_@@_color _ ##1 _tl }

```

```

5503         \tl_gclear:c { g_@@_color _ ##1 _tl }
5504         \pgfusepath { fill }
5505     \end { pgfscope }
5506     }
5507 }
5508 \endpgfpicture
5509 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5510 \cs_new_protected:Npn \@@_color_opacity
5511 {
5512     \peek_meaning:NTF [
5513         { \@@_color_opacity:w }
5514         { \@@_color_opacity:w [ ] }
5515     }

```

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.

```

5516 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5517 {
5518     \tl_clear:N \l_tmpa_tl
5519     \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.

```

5520     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5521     \tl_if_empty:NTF \l_tmpb_tl
5522         { \@declaredcolor }
5523         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5524 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5525 \keys_define:nn { nicematrix / color-opacity }
5526 {
5527     opacity .tl_set:N          = \l_tmpa_tl ,
5528     opacity .value_required:n = true
5529 }

5530 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5531 {
5532     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5533     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5534     \@@_cartesian_path:
5535 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5536 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5537 {
5538     \tl_if_blank:nF { #2 }
5539     {
5540         \@@_add_to_colors_seq:en
5541         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5542         { \@@_cartesian_color:nn { #3 } { - } }
5543     }
5544 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```
5545 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5546 {
5547   \tl_if_blank:nF { #2 }
5548   {
5549     \@@_add_to_colors_seq:en
5550     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5551     { \@@_cartesian_color:nn { - } { #3 } }
5552   }
5553 }
```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```
5554 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5555 {
5556   \tl_if_blank:nF { #2 }
5557   {
5558     \@@_add_to_colors_seq:en
5559     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5560     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5561   }
5562 }
```

The last argument is the radius of the corners of the rectangle.

```
5563 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5564 {
5565   \tl_if_blank:nF { #2 }
5566   {
5567     \@@_add_to_colors_seq:en
5568     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5569     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5570   }
5571 }
```

The last argument is the radius of the corners of the rectangle.

```
5572 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5573 {
5574   \@@_cut_on_hyphen:w #1 \q_stop
5575   \tl_clear_new:N \l_@@_tmpc_tl
5576   \tl_clear_new:N \l_@@_tmpd_tl
5577   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5578   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5579   \@@_cut_on_hyphen:w #2 \q_stop
5580   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5581   \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`.

```
5582   \@@_cartesian_path:n { #3 }
5583 }
```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
5584 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5585 {
5586   \clist_map_inline:nn { #3 }
5587   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5588 }
```

```
5589 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5590 {
5591   \int_step_inline:nn \c@iRow
```

```

5592     {
5593       \int_step_inline:nn \c@jCol
5594       {
5595         \int_if_even:nTF { #####1 + #1 }
5596         { \@@_cellcolor [ #1 ] { #2 } }
5597         { \@@_cellcolor [ #1 ] { #3 } }
5598         { #1 - #####1 }
5599       }
5600     }
5601 }

```

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”.

```

5602 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5603 {
5604   \@@_rectanglecolor [ #1 ] { #2 }
5605   { 1 - 1 }
5606   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5607 }

```

```

5608 \keys_define:nn { nicematrix / rowcolors }
5609 {
5610   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5611   respect-blocks .default:n = true ,
5612   cols .tl_set:N = \l_@@_cols_tl ,
5613   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5614   restart .default:n = true ,
5615   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5616 }

```

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`.

```

5617 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5618 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5619   \group_begin:
5620   \seq_clear_new:N \l_@@_colors_seq
5621   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5622   \tl_clear_new:N \l_@@_cols_tl
5623   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5624   \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).

```

5625   \int_zero_new:N \l_@@_color_int
5626   \int_set_eq:NN \l_@@_color_int \c_one_int
5627   \bool_if:NT \l_@@_respect_blocks_bool
5628   {

```

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`).

```

5629     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5630     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5631     { \@@_not_in_exterior_p:nnnn #1 }
5632   }

```

```

5633 \pgfpicture
5634 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5635 \clist_map_inline:nn { #2 }
5636 {
5637   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5638   \tl_if_in:NnTF \l_tmpa_tl { - }
5639   { \@@_cut_on_hyphen:w ##1 \q_stop }
5640   { \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.

```

5641   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5642   \int_set:Nn \l_@@_color_int
5643   { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5644   \int_zero_new:N \l_@@_tmpc_int
5645   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5646   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5647   {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5648     \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).

```

5649     \bool_if:NT \l_@@_respect_blocks_bool
5650     {
5651       \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5652       { \@@_intersect_our_row_p:nnnnn #####1 }
5653       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5654     }
5655     \tl_set:No \l_@@_rows_tl
5656     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5657     \tl_clear_new:N \l_@@_color_tl
5658     \tl_set:Ne \l_@@_color_tl
5659     {
5660       \@@_color_index:n
5661       {
5662         \int_mod:nn
5663         { \l_@@_color_int - 1 }
5664         { \seq_count:N \l_@@_colors_seq }
5665         + 1
5666       }
5667     }
5668     \tl_if_empty:NF \l_@@_color_tl
5669     {
5670       \@@_add_to_colors_seq:ee
5671       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5672       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5673     }
5674     \int_incr:N \l_@@_color_int
5675     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5676   }
5677 }
5678 \endpgfpicture
5679 \group_end:
5680 }

```

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.

```

5681 \cs_new:Npn \@@_color_index:n #1
5682 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5683 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5684 { \@@_color_index:n { #1 - 1 } }
5685 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5686 }

```

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 curryfication.

```

5687 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5688 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5689 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5690 {
5691 \int_compare:nNnT { #3 } > \l_tmpb_int
5692 { \int_set:Nn \l_tmpb_int { #3 } }
5693 }

```

```

5694 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5695 {
5696 \int_if_zero:nTF { #4 }
5697 \prg_return_false:
5698 {
5699 \int_compare:nNnTF { #2 } > \c@jCol
5700 \prg_return_false:
5701 \prg_return_true:
5702 }
5703 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5704 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5705 {
5706 \int_compare:nNnTF { #1 } > \l_tmpa_int
5707 \prg_return_false:
5708 {
5709 \int_compare:nNnTF \l_tmpa_int > { #3 }
5710 \prg_return_false:
5711 \prg_return_true:
5712 }
5713 }

```

The following command uses two implicit arguments: `\l_@@_rows_tl` and `\l_@@_cols_tl` which are specifications for a set of rows and a set of columns. It creates a path but does *not* fill it. It must be filled by another command after. The argument is the radius of the corners. We define below a command `\@@_cartesian_path:` which corresponds to a value 0 pt for the radius of the corners. This command is, in particular, used in `\@@_rectanglecolor:nnn` (used in `\@@_rectanglecolor`, itself used in `\@@_cellcolor`).

```

5714 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5715 {
5716 \dim_compare:nNnTF { #1 } = \c_zero_dim
5717 {
5718 \bool_if:NTF
5719 \l_@@_nocolor_used_bool
5720 \@@_cartesian_path_normal_ii:
5721 {
5722 \clist_if_empty:NTF \l_@@_corners_cells_clist
5723 { \@@_cartesian_path_normal_i:n { #1 } }
5724 \@@_cartesian_path_normal_ii:

```



```

5725     }
5726   }
5727   { \@@_cartesian_path_normal_i:n { #1 } }
5728 }

```

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.

```

5729 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5730 {
5731   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5732 \clist_map_inline:Nn \l_@@_cols_tl
5733 {
5734   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5735   \tl_if_in:NnTF \l_tmpa_tl { - }
5736     { \@@_cut_on_hyphen:w ##1 \q_stop }
5737     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5738   \tl_if_empty:NTF \l_tmpa_tl
5739     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5740     {
5741       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5742         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5743     }
5744   \tl_if_empty:NTF \l_tmpb_tl
5745     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5746     {
5747       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5748         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5749     }
5750   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5751     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5752   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5753   \@@_qpoint:n { col - \l_tmpa_tl }
5754   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5755     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5756     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5757   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5758   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5759 \clist_map_inline:Nn \l_@@_rows_tl
5760 {
5761   \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5762   \tl_if_in:NnTF \l_tmpa_tl { - }
5763     { \@@_cut_on_hyphen:w #####1 \q_stop }
5764     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5765   \tl_if_empty:NTF \l_tmpa_tl
5766     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5767     {
5768       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5769         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5770     }
5771   \tl_if_empty:NTF \l_tmpb_tl
5772     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5773     {
5774       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5775         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5776     }
5777   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5778     { \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`.

```

5779     \cs_if_exist:cF
5780     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5781     {
5782         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5783         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5784         \@@_qpoint:n { row - \l_tmpa_tl }
5785         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5786         \pgfpathrectanglecorners
5787         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5788         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5789     }
5790 }
5791 }
5792 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5793 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5794 {
5795     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5796     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5797     \clist_map_inline:Nn \l_@@_cols_tl
5798     {
5799         \@@_qpoint:n { col - ##1 }
5800         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5801         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5802         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5803         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5804         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5805     \clist_map_inline:Nn \l_@@_rows_tl
5806     {
5807         \@@_if_in_corner:nF { #####1 - ##1 }
5808         {
5809             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5810             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5811             \@@_qpoint:n { row - #####1 }
5812             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5813             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5814             {
5815                 \pgfpathrectanglecorners
5816                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5817                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5818             }
5819         }
5820     }
5821 }
5822 }

```

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`).

```

5823 \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

```

5824 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5825 {
5826     \bool_set_true:N \l_@@_nocolor_used_bool

```

```

5827 \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5828 \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5829 \clist_map_inline:Nn \l_@@_rows_tl
5830 {
5831   \clist_map_inline:Nn \l_@@_cols_tl
5832   { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ##### } { } }
5833 }
5834 }

```

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.

```

5835 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5836 {
5837   \clist_set_eq:NN \l_tmpa_clist #1
5838   \clist_clear:N #1
5839   \clist_map_inline:Nn \l_tmpa_clist
5840   {
5841     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5842     \tl_if_in:NnTF \l_tmpa_tl { - }
5843     { \@@_cut_on_hyphen:w ##1 \q_stop }
5844     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5845     \bool_lazy_or:nnT
5846     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5847     { \tl_if_blank_p:o \l_tmpa_tl }
5848     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5849     \bool_lazy_or:nnT
5850     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5851     { \tl_if_blank_p:o \l_tmpb_tl }
5852     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5853     \int_compare:nNnT \l_tmpb_tl > #2
5854     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5855     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5856     { \clist_put_right:Nn #1 { ##### } }
5857   }
5858 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5859 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5860 {
5861   \@@_test_color_inside:
5862   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5863   {

```

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 pdf`flatex`).

```

5864     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5865     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5866   }
5867   \ignorespaces
5868 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5869 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5870 {
5871   \@@_test_color_inside:
5872   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5873   {

```

```

5874     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5875     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5876     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5877   }
5878   \ignorespaces
5879 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5880 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5881 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5882 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5883 {
5884   \@@_test_color_inside:
5885   \peek_remove_spaces:n
5886   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5887 }

```

```

5888 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5889 {

```

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`.

```

5890   \seq_gclear:N \g_tmpa_seq
5891   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5892   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5893   \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`.

```

5894   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5895   {
5896     { \int_use:N \c@iRow }
5897     { \exp_not:n { #1 } }
5898     { \exp_not:n { #2 } }
5899     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5900   }
5901 }

```

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`).

```

5902 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5903 {
5904   \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).

```

5905     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5906     {
5907       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5908       {
5909         \@@_rowlistcolors
5910         [ \exp_not:n { #2 } ]
5911         { #1 - \int_eval:n { \c@iRow - 1 } }
5912         { \exp_not:n { #3 } }
5913         [ \exp_not:n { #4 } ]
5914       }
5915     }
5916 }

```

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.

```

5917 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5918   {
5919     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5920     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5921     \seq_gclear:N \g_@@_rowlistcolors_seq
5922   }

5923 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5924   {
5925     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5926     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5927   }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5928 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5929   {

```

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).

```

5930     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5931     {

```

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).

```

5932     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5933     {
5934       \exp_not:N \columncolor [ #1 ]
5935       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5936     }
5937   }
5938 }

```

```

5939 \hook_gput_code:nnn { begindocument } { . }
5940   {
5941     \IfPackageLoadedTF { colortbl }
5942     {
5943       \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5944       \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5945       \cs_new_protected:Npn \@@_revert_colortbl:

```

```

5946     {
5947     \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5948     {
5949     \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5950     \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5951     }
5952     }
5953   }
5954   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5955 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

5956 \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`.

```

5957 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5958 {
5959   \int_if_zero:nTF \l_@@_first_col_int
5960     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5961     {
5962       \int_if_zero:nTF \c@jCol
5963         {
5964           \int_compare:nNnF \c@iRow = { -1 }
5965             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5966         }
5967         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5968     }
5969 }

```

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.

```

5970 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5971 {
5972   \int_if_zero:nF \c@iRow
5973     {
5974       \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5975         {
5976           \int_compare:nNnT \c@jCol > \c_zero_int
5977             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5978         }
5979     }
5980 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5981 \keys_define:nn { nicematrix / Rules }
5982 {
5983   position .int_set:N = \l_@@_position_int ,
5984   position .value_required:n = true ,
5985   start .int_set:N = \l_@@_start_int ,
5986   end .code:n =
5987     \bool_lazy_or:nnTF
5988     { \tl_if_empty_p:n { #1 } }
5989     { \str_if_eq_p:ee { #1 } { last } }
5990     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5991     { \int_set:Nn \l_@@_end_int { #1 } }
5992 }
```

It’s possible that the rule won’t be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by `\@@_vline_ii:` and `\@@_hline_ii:`. Those commands use the following set of keys.

```
5993 \keys_define:nn { nicematrix / RulesBis }
5994 {
5995   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5996   multiplicity .initial:n = 1 ,
5997   dotted .bool_set:N = \l_@@_dotted_bool ,
5998   dotted .initial:n = false ,
5999   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]`.

```
6000   color .code:n =
6001     \@@_set_CT@arc@:n { #1 }
6002     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6003   color .value_required:n = true ,
6004   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6005   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.

```
6006   tikz .code:n =
6007     \IfPackageLoadedTF { tikz }
6008     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6009     { \@@_error:n { tikz-without-tikz } } ,
6010   tikz .value_required:n = true ,
6011   total-width .dim_set:N = \l_@@_rule_width_dim ,
6012   total-width .value_required:n = true ,
6013   width .meta:n = { total-width = #1 } ,
6014   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6015 }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of *key=value* pairs.

```
6016 \cs_new_protected:Npn \@@_vline:n #1
6017 {
```

The group is for the options.

```
6018   \group_begin:
6019   \int_set_eq:NN \l_@@_end_int \c@iRow
6020   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of |c|c|c| but only two columns used).

```
6021   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6022     \@@_vline_i:
6023   \group_end:
6024 }

6025 \cs_new_protected:Npn \@@_vline_i:
6026 {
```

`\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`.

```
6027   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6028   \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6029     \l_tmpa_tl
6030   {
```

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.

```
6031     \bool_gset_true:N \g_tmpa_bool
6032     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6033       { \@@_test_vline_in_block:nnnn #1 }
6034     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6035       { \@@_test_vline_in_block:nnnn #1 }
6036     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6037       { \@@_test_vline_in_stroken_block:nnnn #1 }
6038     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6039     \bool_if:NTF \g_tmpa_bool
6040     {
6041       \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.

```
6042       { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6043     }
6044     {
6045       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6046       {
6047         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6048         \@@_vline_ii:
6049         \int_zero:N \l_@@_local_start_int
6050       }
6051     }
6052   }
6053   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6054   {
6055     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6056     \@@_vline_ii:
6057   }
6058 }
```

```
6059 \cs_new_protected:Npn \@@_test_in_corner_v:
6060 {
6061   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6062   {
6063     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
```



```

6064     { \bool_set_false:N \g_tmpa_bool }
6065   }
6066   {
6067     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6068     {
6069       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6070       { \bool_set_false:N \g_tmpa_bool }
6071       {
6072         \@@_if_in_corner:nT
6073         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6074         { \bool_set_false:N \g_tmpa_bool }
6075       }
6076     }
6077   }
6078 }

```

```

6079 \cs_new_protected:Npn \@@_vline_ii:
6080 {
6081   \tl_clear:N \l_@@_tikz_rule_tl
6082   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6083   \bool_if:NTF \l_@@_dotted_bool
6084     \@@_vline_iv:
6085     {
6086       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6087       \@@_vline_iii:
6088       \@@_vline_v:
6089     }
6090 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6091 \cs_new_protected:Npn \@@_vline_iii:
6092 {
6093   \pgfpicture
6094   \pgfrememberpicturepositiononpagetrue
6095   \pgf@relevantforpicturesizefalse
6096   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6097   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6098   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6099   \dim_set:Nn \l_tmpb_dim
6100   {
6101     \pgf@x
6102     - 0.5 \l_@@_rule_width_dim
6103     +
6104     ( \arrayrulewidth * \l_@@_multiplicity_int
6105       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6106   }
6107   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6108   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6109   \bool_lazy_all:nT
6110   {
6111     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6112     { \cs_if_exist_p:N \CT@drsc@ }
6113     { ! \tl_if_blank_p:o \CT@drsc@ }
6114   }
6115   {
6116     \group_begin:
6117     \CT@drsc@
6118     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6119     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6120     \dim_set:Nn \l_@@_tmpd_dim
6121     {
6122       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )

```

```

6123     * ( \l_@@_multiplicity_int - 1 )
6124   }
6125   \pgfpathrectanglecorners
6126     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6127     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6128   \pgfusepath { fill }
6129   \group_end:
6130 }
6131 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6132 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6133 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6134 {
6135   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6136   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6137   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6138   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6139 }
6140 \CT@arc@
6141 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6142 \pgfsetrectcap
6143 \pgfusepathqstroke
6144 \endpgfpicture
6145 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6146 \cs_new_protected:Npn \@@_vline_iv:
6147 {
6148   \pgfpicture
6149   \pgfrememberpicturepositiononpagetrue
6150   \pgf@relevantforpicturesizefalse
6151   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6152   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6153   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6154   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6155   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6156   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6157   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6158   \CT@arc@
6159   \@@_draw_line:
6160   \endpgfpicture
6161 }

```

The following code is for the case when the user uses the key `tikz`.

```

6162 \cs_new_protected:Npn \@@_vline_v:
6163 {
6164   \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).

```

6165   \CT@arc@
6166   \tl_if_empty:NF \l_@@_rule_color_tl
6167   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6168   \pgfrememberpicturepositiononpagetrue
6169   \pgf@relevantforpicturesizefalse
6170   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6171   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6172   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6173   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6174   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6175   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6176   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6177   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }

```

```

6178     ( \l_tmpb_dim , \l_tmpa_dim ) --
6179     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6180 \end { tikzpicture }
6181 }

```

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).

```

6182 \cs_new_protected:Npn \@@_draw_vlines:
6183 {
6184   \int_step_inline:nnn
6185     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6186     {
6187       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6188         \c@jCol
6189         { \int_eval:n { \c@jCol + 1 } }
6190     }
6191     {
6192       \tl_if_eq:NnF \l_@@_vlines_clist \c_@@_all_tl
6193         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6194         { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6195     }
6196 }

```

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}`.

```

6197 \cs_new_protected:Npn \@@_hline:n #1
6198 {

```

The group is for the options.

```

6199   \group_begin:
6200   \int_zero_new:N \l_@@_end_int
6201   \int_set_eq:NN \l_@@_end_int \c@jCol
6202   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6203   \@@_hline_i:
6204   \group_end:
6205 }

6206 \cs_new_protected:Npn \@@_hline_i:
6207 {
6208   \int_zero_new:N \l_@@_local_start_int
6209   \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`.

```

6210   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6211   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6212     \l_tmpb_tl
6213     {

```

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.

```

6214       \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6215       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6216         { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6217 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6218 { \@@_test_hline_in_block:nnnnn ##1 }
6219 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6220 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6221 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6222 \bool_if:NTF \g_tmpa_bool
6223 {
6224 \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.

```

6225 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6226 }
6227 {
6228 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6229 {
6230 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6231 \@@_hline_ii:
6232 \int_zero:N \l_@@_local_start_int
6233 }
6234 }
6235 }
6236 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6237 {
6238 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6239 \@@_hline_ii:
6240 }
6241 }

```

```

6242 \cs_new_protected:Npn \@@_test_in_corner_h:
6243 {
6244 \int_compare:nNnTF \l_tmpa_tl = { \c_iRow + 1 }
6245 {
6246 \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6247 { \bool_set_false:N \g_tmpa_bool }
6248 }
6249 {
6250 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6251 {
6252 \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6253 { \bool_set_false:N \g_tmpa_bool }
6254 {
6255 \@@_if_in_corner:nT
6256 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6257 { \bool_set_false:N \g_tmpa_bool }
6258 }
6259 }
6260 }
6261 }

```

```

6262 \cs_new_protected:Npn \@@_hline_ii:
6263 {
6264 \tl_clear:N \l_@@_tikz_rule_tl
6265 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6266 \bool_if:NTF \l_@@_dotted_bool
6267 \@@_hline_iv:
6268 {
6269 \tl_if_empty:NTF \l_@@_tikz_rule_tl
6270 \@@_hline_iii:
6271 \@@_hline_v:
6272 }
6273 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6274 \cs_new_protected:Npn \@@_hline_iii:
6275 {
6276   \pgfpicture
6277   \pgfrememberpicturepositiononpagetrue
6278   \pgf@relevantforpicturesizefalse
6279   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6280   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6281   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6282   \dim_set:Nn \l_tmpb_dim
6283     {
6284       \pgf@y
6285       - 0.5 \l_@@_rule_width_dim
6286       +
6287       ( \arrayrulewidth * \l_@@_multiplicity_int
6288         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6289     }
6290   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6291   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6292   \bool_lazy_all:nT
6293     {
6294       { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6295       { \cs_if_exist_p:N \CT@drsc@ }
6296       { ! \tl_if_blank_p:o \CT@drsc@ }
6297     }
6298     {
6299       \group_begin:
6300       \CT@drsc@
6301       \dim_set:Nn \l_@@_tmpd_dim
6302         {
6303           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6304           * ( \l_@@_multiplicity_int - 1 )
6305         }
6306       \pgfpathrectanglecorners
6307         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6308         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6309       \pgfusepathqfill
6310       \group_end:
6311     }
6312   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6313   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6314   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6315     {
6316       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6317       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6318       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6319       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6320     }
6321   \CT@arc@
6322   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6323   \pgfsetrectcap
6324   \pgfusepathqstroke
6325   \endpgfpicture
6326 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6327 \cs_new_protected:Npn \@@_hline_iv:
6328   {
6329     \pgfpicture
6330     \pgfrememberpicturepositiononpagetrue
6331     \pgf@relevantforpicturesizefalse
6332     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6333     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6334     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6335     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6336     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6337     \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6338     {
6339       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6340       \bool_if:NF \g_@@_delims_bool
6341       { \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.

```

6342     \tl_if_eq:NnF \g_@@_left_delim_tl (
6343       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6344     )
6345     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6346     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6347     \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6348     {
6349       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6350       \bool_if:NF \g_@@_delims_bool
6351       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6352       \tl_if_eq:NnF \g_@@_right_delim_tl )
6353       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6354     }
6355     \CT@arc@
6356     \@@_draw_line:
6357     \endpgfpicture
6358   }

```

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`).

```

6359 \cs_new_protected:Npn \@@_hline_v:
6360   {
6361     \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).

```

6362     \CT@arc@
6363     \tl_if_empty:NF \l_@@_rule_color_tl

```

```

6364     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6365 \pgfrememberpicturepositiononpagetrue
6366 \pgf@relevantforpicturesizefalse
6367 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6368 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6369 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6370 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6371 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6372 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6373 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6374 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6375   ( \l_tmpa_dim , \l_tmpb_dim ) --
6376   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6377 \end { tikzpicture }
6378 }

```

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).

```

6379 \cs_new_protected:Npn \@@_draw_hlines:
6380 {
6381   \int_step_inline:nnn
6382     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6383     {
6384       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6385         \c@iRow
6386         { \int_eval:n { \c@iRow + 1 } }
6387     }
6388     {
6389       \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
6390       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6391       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6392     }
6393 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6394 \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.

```

6395 \cs_set:Npn \@@_Hline_i:n #1
6396 {
6397   \peek_remove_spaces:n
6398   {
6399     \peek_meaning:NTF \Hline
6400     { \@@_Hline_ii:nn { #1 + 1 } }
6401     { \@@_Hline_iii:n { #1 } }
6402   }
6403 }
6404 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6405 \cs_set:Npn \@@_Hline_iii:n #1
6406 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6407 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6408 {
6409   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6410   \skip_vertical:N \l_@@_rule_width_dim
6411   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6412   {
6413     \@@_hline:n
6414     {
6415       multiplicity = #1 ,
6416       position = \int_eval:n { \c@iRow + 1 } ,

```

```

6417         total-width = \dim_use:N \l_@@_rule_width_dim ,
6418         #2
6419     }
6420 }
6421 \egroup
6422 }

```

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`).

```

6423 \cs_new_protected:Npn \@@_custom_line:n #1
6424 {
6425     \str_clear_new:N \l_@@_command_str
6426     \str_clear_new:N \l_@@_ccommand_str
6427     \str_clear_new:N \l_@@_letter_str
6428     \tl_clear_new:N \l_@@_other_keys_tl
6429     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6430     \bool_lazy_all:nTF
6431     {
6432         { \str_if_empty_p:N \l_@@_letter_str }
6433         { \str_if_empty_p:N \l_@@_command_str }
6434         { \str_if_empty_p:N \l_@@_ccommand_str }
6435     }
6436     { \@@_error:n { No~letter~and~no~command } }
6437     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6438 }
6439 \keys_define:nn { nicematrix / custom-line }
6440 {
6441     letter .str_set:N = \l_@@_letter_str ,
6442     letter .value_required:n = true ,
6443     command .str_set:N = \l_@@_command_str ,
6444     command .value_required:n = true ,
6445     ccommand .str_set:N = \l_@@_ccommand_str ,
6446     ccommand .value_required:n = true ,
6447 }
6448 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6449 \cs_new_protected:Npn \@@_custom_line_i:n #1
6450 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6451     \bool_set_false:N \l_@@_tikz_rule_bool
6452     \bool_set_false:N \l_@@_dotted_rule_bool
6453     \bool_set_false:N \l_@@_color_bool
6454     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6455     \bool_if:NT \l_@@_tikz_rule_bool
6456     {
6457         \IfPackageLoadedF { tikz }
6458         { \@@_error:n { tikz-in-custom-line-without-tikz } }
6459         \bool_if:NT \l_@@_color_bool
6460         { \@@_error:n { color-in-custom-line-with-tikz } }
6461     }

```



```

6462 \bool_if:NT \l_@@_dotted_rule_bool
6463 {
6464   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6465   { \@@_error:n { key~multiplicity~with~dotted } }
6466 }
6467 \str_if_empty:NF \l_@@_letter_str
6468 {
6469   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6470   { \@@_error:n { Several~letters } }
6471   {
6472     \tl_if_in:NoTF
6473     \c_@@_forbidden_letters_str
6474     \l_@@_letter_str
6475     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6476   }

```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```

6477   \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6478   { \@@_v_custom_line:n { #1 } }
6479 }
6480 }
6481 }
6482 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6483 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6484 }
6485 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6486 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|() []!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{nicematrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6487 \keys_define:nn { nicematrix / custom-line-bis }
6488 {
6489   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6490   multiplicity .initial:n = 1 ,
6491   multiplicity .value_required:n = true ,
6492   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6493   color .value_required:n = true ,
6494   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6495   tikz .value_required:n = true ,
6496   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6497   dotted .value_forbidden:n = true ,
6498   total-width .code:n = { } ,
6499   total-width .value_required:n = true ,
6500   width .code:n = { } ,
6501   width .value_required:n = true ,
6502   sep-color .code:n = { } ,
6503   sep-color .value_required:n = true ,
6504   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6505 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6506 \bool_new:N \l_@@_dotted_rule_bool
6507 \bool_new:N \l_@@_tikz_rule_bool
6508 \bool_new:N \l_@@_color_bool

```

The following keys are used to determine the total width of the line (including the spaces on both sides of the line). The key `width` is deprecated and has been replaced by the key `total-width`.

```

6509 \keys_define:n#n { nicematrix / custom-line-width }
6510 {
6511   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6512   multiplicity .initial:n = 1 ,
6513   multiplicity .value_required:n = true ,
6514   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6515   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6516                   \bool_set_true:N \l_@@_total_width_bool ,
6517   total-width .value_required:n = true ,
6518   width .meta:n = { total-width = #1 } ,
6519   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6520 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. #1 is the whole set of keys to pass to the command \@@_hline:n (which is in the internal \CodeAfter).

```

6521 \cs_new_protected:Npn \@@_h_custom_line:n #1
6522 {

```

We use \cs_set:cpn and not \cs_new:cpn because we want a local definition. Moreover, the command must *not* be protected since it begins with \noalign (which is in \Hline).

```

6523   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6524   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6525 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule on only some of the columns of the array (hence the letter c as in \cline). #1 is the whole set of keys to pass to the command \@@_hline:n (which is in the internal \CodeAfter).

```

6526 \cs_new_protected:Npn \@@_c_custom_line:n #1
6527 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6528   \exp_args:Nc \NewExpandableDocumentCommand
6529   { nicematrix - \l_@@_ccommand_str }
6530   { 0 { } m }
6531   {
6532     \noalign
6533     {
6534       \@@_compute_rule_width:n { #1 , ##1 }
6535       \skip_vertical:n { \l_@@_rule_width_dim }
6536       \clist_map_inline:nn
6537       { ##2 }
6538       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6539     }
6540   }
6541   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6542 }

```

The first argument is the list of key-value pairs characteristic of the line. The second argument is the specification of columns for the \cline with the syntax *a-b*.

```

6543 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6544 {
6545   \tl_if_in:nnTF { #2 } { - }
6546   { \@@_cut_on_hyphen:w #2 \q_stop }
6547   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6548   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6549   {
6550     \@@_hline:n
6551     {
6552       #1 ,
6553       start = \l_tmpa_tl ,

```

```

6554         end = \l_tmpb_tl ,
6555         position = \int_eval:n { \c@iRow + 1 } ,
6556         total-width = \dim_use:N \l_@@_rule_width_dim
6557     }
6558 }
6559 }
6560 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6561 {
6562     \bool_set_false:N \l_@@_tikz_rule_bool
6563     \bool_set_false:N \l_@@_total_width_bool
6564     \bool_set_false:N \l_@@_dotted_rule_bool
6565     \keys_set_known:nm { nicematrix / custom-line-width } { #1 }
6566     \bool_if:NF \l_@@_total_width_bool
6567     {
6568         \bool_if:NTF \l_@@_dotted_rule_bool
6569         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6570         {
6571             \bool_if:NF \l_@@_tikz_rule_bool
6572             {
6573                 \dim_set:Nn \l_@@_rule_width_dim
6574                 {
6575                     \arrayrulewidth * \l_@@_multiplicity_int
6576                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6577                 }
6578             }
6579         }
6580     }
6581 }
6582 \cs_new_protected:Npn \@@_v_custom_line:n #1
6583 {
6584     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6585     \tl_gput_right:Ne \g_@@_array_preamble_tl
6586     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6587     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6588     {
6589         \@@_vline:n
6590         {
6591             #1 ,
6592             position = \int_eval:n { \c@jCol + 1 } ,
6593             total-width = \dim_use:N \l_@@_rule_width_dim
6594         }
6595     }
6596     \@@_rec_preamble:n
6597 }
6598 \@@_custom_line:n
6599 { letter = : , command = hdottedline , ccommand = cdottedline , dotted }

```

The key hvlines

The following command tests whether the current position in the array (given by `\l_tmpa_tl` for the row and `\l_tmpb_tl` for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments #1, #2, #3 and #4. If this rule would be in the block (it must not be drawn), the boolean `\l_tmpa_bool` is set to false.

```

6600 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6601 {
6602     \int_compare:nNnT \l_tmpa_tl > { #1 }
6603     {
6604         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6605         {
6606             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }

```

```

6607         {
6608             \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6609             { \bool_gset_false:N \g_tmpa_bool }
6610         }
6611     }
6612 }
6613 }

```

The same for vertical rules.

```

6614 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6615 {
6616     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6617     {
6618         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6619         {
6620             \int_compare:nNnT \l_tmpb_tl > { #2 }
6621             {
6622                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6623                 { \bool_gset_false:N \g_tmpa_bool }
6624             }
6625         }
6626     }
6627 }
6628 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6629 {
6630     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6631     {
6632         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6633         {
6634             \int_compare:nNnTF \l_tmpa_tl = { #1 }
6635             { \bool_gset_false:N \g_tmpa_bool }
6636             {
6637                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6638                 { \bool_gset_false:N \g_tmpa_bool }
6639             }
6640         }
6641     }
6642 }
6643 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6644 {
6645     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6646     {
6647         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6648         {
6649             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6650             { \bool_gset_false:N \g_tmpa_bool }
6651             {
6652                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6653                 { \bool_gset_false:N \g_tmpa_bool }
6654             }
6655         }
6656     }
6657 }

```

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.

```

6658 \cs_new_protected:Npn \@@_compute_corners:
6659 {
6660   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6661     { \@@_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`).

```

6662   \clist_clear:N \l_@@_corners_cells_clist
6663   \clist_map_inline:Nn \l_@@_corners_clist
6664     {
6665     \str_case:nnF { ##1 }
6666       {
6667         { NW }
6668         { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6669         { NE }
6670         { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6671         { SW }
6672         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6673         { SE }
6674         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6675       }
6676     { \@@_error:nn { bad~corner } { ##1 } }
6677   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6678   \clist_if_empty:NF \l_@@_corners_cells_clist
6679   {

```

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.

```

6680     \tl_gput_right:Ne \g_@@_aux_tl
6681     {
6682       \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6683         { \l_@@_corners_cells_clist }
6684     }
6685   }
6686 }

```

```

6687 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6688 {
6689   \int_step_inline:nnn { #1 } { #3 }
6690   {
6691     \int_step_inline:nnn { #2 } { #4 }
6692     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6693   }
6694 }

```

```

6695 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6696 {
6697   \cs_if_exist:cTF
6698     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6699     \prg_return_true:
6700     \prg_return_false:
6701 }

```

“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.

```
6702 \cs_new_protected:Npn \@@_compute_a_corner:nnnnn #1 #2 #3 #4 #5 #6
6703 {
```

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.

```
6704 \bool_set_false:N \l_tmpa_bool
6705 \int_zero_new:N \l_@@_last_empty_row_int
6706 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6707 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6708 {
6709   \bool_lazy_or:nnTF
6710   {
6711     \cs_if_exist_p:c
6712     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6713   }
6714   { \@@_if_in_block_p:nn { ##1 } { #2 } }
6715   { \bool_set_true:N \l_tmpa_bool }
6716   {
6717     \bool_if:NF \l_tmpa_bool
6718     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6719   }
6720 }
```

Now, you determine the last empty cell in the row of number 1.

```
6721 \bool_set_false:N \l_tmpa_bool
6722 \int_zero_new:N \l_@@_last_empty_column_int
6723 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6724 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6725 {
6726   \bool_lazy_or:nnTF
6727   {
6728     \cs_if_exist_p:c
6729     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6730   }
6731   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6732   { \bool_set_true:N \l_tmpa_bool }
6733   {
6734     \bool_if:NF \l_tmpa_bool
6735     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6736   }
6737 }
```

Now, we loop over the rows.

```
6738 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6739 {
```

We treat the row number `##1` with another loop.

```
6740 \bool_set_false:N \l_tmpa_bool
6741 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6742 {
6743   \bool_lazy_or:nnTF
6744   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6745   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6746   { \bool_set_true:N \l_tmpa_bool }
6747   {
6748     \bool_if:NF \l_tmpa_bool
```

```

6749         {
6750             \int_set:Nn \l_@@_last_empty_column_int { ##### }
6751             \clist_put_right:Nn
6752                 \l_@@_corners_cells_clist
6753                 { ##1 - ##### }
6754             \cs_set_nopar:cpn { @@ _ corner _ ##1 - ##### } { }
6755         }
6756     }
6757 }
6758 }
6759 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6760 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6761 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

24 The environment `{NiceMatrixBlock}`

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6762 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6763 \keys_define:nn { nicematrix / NiceMatrixBlock }
6764 {
6765     auto-columns-width .code:n =
6766     {
6767         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6768         \dim_gzero_new:N \g_@@_max_cell_width_dim
6769         \bool_set_true:N \l_@@_auto_columns_width_bool
6770     }
6771 }

6772 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6773 {
6774     \int_gincr:N \g_@@_NiceMatrixBlock_int
6775     \dim_zero:N \l_@@_columns_width_dim
6776     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6777     \bool_if:NT \l_@@_block_auto_columns_width_bool
6778     {
6779         \cs_if_exist:cT
6780             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6781         {
6782             \dim_set:Nn \l_@@_columns_width_dim
6783             {
6784                 \use:c
6785                     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6786             }
6787         }
6788     }
6789 }

```

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`).

```
6790 {
6791   \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.

```
6792   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6793   {
6794     \bool_if:NT \l_@@_block_auto_columns_width_bool
6795     {
6796       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6797       \iow_shipout:Ne \@mainaux
6798       {
6799         \cs_gset:cpn
6800         { @@ _ 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.

```
6801         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6802       }
6803       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6804     }
6805   }
6806   \ignorespacesafterend
6807 }
```

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).

```
6808 \cs_new_protected:Npn \@@_create_extra_nodes:
6809 {
6810   \bool_if:nTF \l_@@_medium_nodes_bool
6811   {
6812     \bool_if:NTF \l_@@_large_nodes_bool
6813     \@@_create_medium_and_large_nodes:
6814     \@@_create_medium_nodes:
6815   }
6816   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6817 }
```

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`.

```

6818 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6819 {
6820   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6821   {
6822     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6823     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6824     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6825     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6826   }
6827   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6828   {
6829     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6830     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6831     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6832     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6833   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6834   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6835   {
6836     \int_step_variable:nnNn
6837     \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.

```

6838     {
6839       \cs_if_exist:cT
6840       { 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`.

```

6841     {
6842       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6843       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6844       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6845       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6846       {
6847         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6848         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6849       }

```

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`.

```

6850       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6851       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6852       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6853       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6854       {
6855         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6856         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6857       }
6858     }
6859   }
6860 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6861   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6862   {
6863     \dim_compare:nNnT
6864     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim

```

```

6865     {
6866     \@@_qpoint:n { row - \@@_i: - base }
6867     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6868     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6869     }
6870   }
6871 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6872 {
6873   \dim_compare:nNnT
6874   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6875   {
6876     \@@_qpoint:n { col - \@@_j: }
6877     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6878     \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6879   }
6880 }
6881 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6882 \cs_new_protected:Npn \@@_create_medium_nodes:
6883 {
6884   \pgfpicture
6885   \pgfrememberpicturepositiononpagetrue
6886   \pgf@relevantforpicturesizefalse
6887   \@@_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”.

```

6888     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6889     \@@_create_nodes:
6890     \endpgfpicture
6891 }

```

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:`.

```

6892 \cs_new_protected:Npn \@@_create_large_nodes:
6893 {
6894   \pgfpicture
6895   \pgfrememberpicturepositiononpagetrue
6896   \pgf@relevantforpicturesizefalse
6897   \@@_computations_for_medium_nodes:
6898   \@@_computations_for_large_nodes:
6899   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6900   \@@_create_nodes:
6901   \endpgfpicture
6902 }
6903 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6904 {
6905   \pgfpicture
6906   \pgfrememberpicturepositiononpagetrue
6907   \pgf@relevantforpicturesizefalse
6908   \@@_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”.

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6909     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6910     \@@_create_nodes:
6911     \@@_computations_for_large_nodes:
6912     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6913     \@@_create_nodes:
6914     \endpgfpicture
6915 }

```

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.

```

6916 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6917 {
6918     \int_set_eq:NN \l_@@_first_row_int \c_one_int
6919     \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`.

```

6920     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6921     {
6922         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6923         {
6924             (
6925                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6926                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6927             )
6928             / 2
6929         }
6930         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6931         { l_@@_row_\@@_i: _ min_dim }
6932     }
6933     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6934     {
6935         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6936         {
6937             (
6938                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6939                 \dim_use:c
6940                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6941             )
6942             / 2
6943         }
6944         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6945         { l_@@_column _ \@@_j: _ max _ dim }
6946     }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6947     \dim_sub:cn
6948     { l_@@_column _ 1 _ min _ dim }
6949     \l_@@_left_margin_dim
6950     \dim_add:cn
6951     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6952     \l_@@_right_margin_dim
6953 }

```

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`).

```

6954 \cs_new_protected:Npn \@@_create_nodes:
6955 {

```

```

6956 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6957 {
6958   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6959   {

```

We draw the rectangular node for the cell ($\backslash\@@_i-\backslash\@@_j$).

```

6960     \@@_pgf_rect_node:nnnnn
6961     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6962     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6963     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6964     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6965     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6966     \str_if_empty:NF \l_@@_name_str
6967     {
6968       \pgfnodealias
6969       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6970       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6971     }
6972   }
6973 }

```

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 .

```

6974   \seq_map_pairwise_function:NNN
6975   \g_@@_multicolumn_cells_seq
6976   \g_@@_multicolumn_sizes_seq
6977   \@@_node_for_multicolumn:nn
6978 }

6979 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6980 {
6981   \cs_set_nopar:Npn \@@_i: { #1 }
6982   \cs_set_nopar:Npn \@@_j: { #2 }
6983 }

```

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”).

```

6984 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6985 {
6986   \@@_extract_coords_values: #1 \q_stop
6987   \@@_pgf_rect_node:nnnnn
6988   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6989   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6990   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6991   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6992   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6993   \str_if_empty:NF \l_@@_name_str
6994   {
6995     \pgfnodealias
6996     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6997     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6998   }
6999 }

```

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).

```

7000 \keys_define:nn { nicematrix / Block / FirstPass }
7001   {
7002     j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7003         \bool_set_true:N \l_@@_p_block_bool ,
7004     j .value_forbidden:n = true ,
7005     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7006     l .value_forbidden:n = true ,
7007     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7008     r .value_forbidden:n = true ,
7009     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7010     c .value_forbidden:n = true ,
7011     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7012     L .value_forbidden:n = true ,
7013     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7014     R .value_forbidden:n = true ,
7015     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7016     C .value_forbidden:n = true ,
7017     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7018     t .value_forbidden:n = true ,
7019     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7020     T .value_forbidden:n = true ,
7021     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7022     b .value_forbidden:n = true ,
7023     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7024     B .value_forbidden:n = true ,
7025     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7026     m .value_forbidden:n = true ,
7027     v-center .meta:n = m ,
7028     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7029     p .value_forbidden:n = true ,
7030     color .code:n =
7031         \@@_color:n { #1 }
7032         \tl_set_rescan:Nnn
7033             \l_@@_draw_tl
7034             { \char_set_catcode_other:N ! }
7035             { #1 } ,
7036     color .value_required:n = true ,
7037     respect-arraystretch .code:n =
7038         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7039     respect-arraystretch .value_forbidden:n = true ,
7040   }

```

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.

```

7041 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7042 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7043   {

```

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).

```

7044     \peek_remove_spaces:n
7045     {
7046         \tl_if_blank:nTF { #2 }
7047             { \@@_Block_ii:nnnn \c_one_int \c_one_int }
7048             {
7049                 \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7050                 \@@_Block_i_czech \@@_Block_i

```

```

7051         #2 \q_stop
7052     }
7053     { #1 } { #3 } { #4 }
7054 }
7055 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7056 \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`.

```

7057 {
7058   \char_set_catcode_active:N -
7059   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7060 }

```

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.

```

7061 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7062 {

```

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).

```

7063   \bool_lazy_or:nnTF
7064     { \tl_if_blank_p:n { #1 } }
7065     { \str_if_eq_p:ee { * } { #1 } }
7066     { \int_set:Nn \l_tmpa_int { 100 } }
7067     { \int_set:Nn \l_tmpa_int { #1 } }
7068   \bool_lazy_or:nnTF
7069     { \tl_if_blank_p:n { #2 } }
7070     { \str_if_eq_p:ee { * } { #2 } }
7071     { \int_set:Nn \l_tmpb_int { 100 } }
7072     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7073   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7074     {
7075       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7076         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7077         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7078     }
7079     { \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.

```

7080   \keys_set_known:nm { nicematrix / Block / FirstPass } { #3 }
7081   \tl_set:Ne \l_tmpa_tl
7082   {
7083     { \int_use:N \c@iRow }
7084     { \int_use:N \c@jCol }
7085     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7086     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7087   }

```

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).

```
7088 \bool_set_false:N \l_tmpa_bool
7089 \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```
7090 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7091 \bool_case:nF
7092 {
7093 \l_tmpa_bool { \@@_Block_vii:eennn }
7094 \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.

```
7095 \l_@@_X_bool { \@@_Block_v:eennn }
7096 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7097 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7098 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7099 }
7100 { \@@_Block_v:eennn }
7101 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7102 }
```

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.

```
7103 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7104 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7105 {
7106 \int_gincr:N \g_@@_block_box_int
7107 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7108 {
7109 \tl_gput_right:Ne \g_@@_pre_code_after_tl
7110 {
7111 \@@_actually_diagbox:nnnnnn
7112 { \int_use:N \c@iRow }
7113 { \int_use:N \c@jCol }
7114 { \int_eval:n { \c@iRow + #1 - 1 } }
7115 { \int_eval:n { \c@jCol + #2 - 1 } }
7116 { \g_@@_row_style_tl \exp_not:n { ##1 } }
7117 { \g_@@_row_style_tl \exp_not:n { ##2 } }
7118 }
7119 }
7120 \box_gclear_new:c
7121 { \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!

```

7122   \hbox_gset:cn
7123   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7124   {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current`: (in order to use `\color_ensure_current`: safely, you should load `l3backend` before the `\documentclass` with `\RequirePackage{expl3}`).

```

7125   \tl_if_empty:NTF \l_@@_color_tl
7126   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7127   { \@@_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`.

```

7128   \int_compare:nNnT { #1 } = \c_one_int
7129   {
7130     \int_if_zero:nTF \c@iRow
7131     {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7132   \cs_set_eq:NN \Block \@@_NullBlock:
7133   \l_@@_code_for_first_row_tl
7134   }
7135   {
7136     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7137     {
7138       \cs_set_eq:NN \Block \@@_NullBlock:
7139       \l_@@_code_for_last_row_tl
7140     }
7141   }
7142   \g_@@_row_style_tl
7143   }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7144   \@@_reset_arraystretch:
7145   \dim_zero:N \extrarowheight

```

`#4` is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7146   #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.).

```
7147     \@@_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}`.

```
7148     \bool_if:NTF \l_@@_tabular_bool
7149     {
7150         \bool_lazy_all:nTF
7151         {
7152             { \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.

```
7153         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7154         { ! \g_@@_rotate_bool }
7155     }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7156     {
7157         \use:e
7158     }
```

The `\exp_not:N` is mandatory before `\begin`.

```
7159         \exp_not:N \begin { minipage }%
7160         [ \str_lowercase:o \l_@@_vpos_block_str ]
7161         { \l_@@_col_width_dim }
7162         \str_case:on \l_@@_hpos_block_str
7163         { c \centering r \raggedleft l \raggedright }
7164     }
7165     #5
7166     \end { minipage }
7167 }
```

In the other cases, we use a `{tabular}`.

```
7168     {
7169         \use:e
7170     {
7171         \exp_not:N \begin { tabular }%
7172         [ \str_lowercase:o \l_@@_vpos_block_str ]
7173         { @ { } \l_@@_hpos_block_str @ { } }
7174     }
7175     #5
7176     \end { tabular }
7177 }
7178 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7179     {
7180         \c_math_toggle_token
7181         \use:e
7182     {
7183         \exp_not:N \begin { array }%
7184         [ \str_lowercase:o \l_@@_vpos_block_str ]
7185         { @ { } \l_@@_hpos_block_str @ { } }
7186     }
7187     #5
7188     \end { array }
7189     \c_math_toggle_token
7190 }
7191 }
```

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).

```
7192   \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.

```
7193   \int_compare:nNnT { #2 } = \c_one_int
7194     {
7195       \dim_gset:Nn \g_@@_blocks_wd_dim
7196         {
7197           \dim_max:nn
7198             \g_@@_blocks_wd_dim
7199             {
7200               \box_wd:c
7201                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7202             }
7203         }
7204     }
```

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.

```
7205   \bool_lazy_and:nnT
7206     { \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.

```
7207     { \str_if_empty_p:N \l_@@_vpos_block_str }
7208     {
7209       \dim_gset:Nn \g_@@_blocks_ht_dim
7210         {
7211           \dim_max:nn
7212             \g_@@_blocks_ht_dim
7213             {
7214               \box_ht:c
7215                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7216             }
7217         }
7218       \dim_gset:Nn \g_@@_blocks_dp_dim
7219         {
7220           \dim_max:nn
7221             \g_@@_blocks_dp_dim
7222             {
7223               \box_dp:c
7224                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7225             }
7226         }
7227     }
7228   \seq_gput_right:Ne \g_@@_blocks_seq
7229     {
7230       \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.

```
7231     {
7232       \exp_not:n { #3 } ,
7233       \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
7234       \bool_if:NT \g_@@_rotate_bool
```

```

7235     {
7236         \bool_if:NTF \g_@@_rotate_c_bool
7237         { m }
7238         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7239     }
7240 }
7241 {
7242     \box_use_drop:c
7243     { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7244 }
7245 }
7246 \bool_set_false:N \g_@@_rotate_c_bool
7247 }

7248 \cs_new:Npn \@@_adjust_hpos_rotate:
7249 {
7250     \bool_if:NT \g_@@_rotate_bool
7251     {
7252         \str_set:Ne \l_@@_hpos_block_str
7253         {
7254             \bool_if:NTF \g_@@_rotate_c_bool
7255             { c }
7256             {
7257                 \str_case:onF \l_@@_vpos_block_str
7258                 { b l B l t r T r }
7259                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7260             }
7261         }
7262     }
7263 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7264 \cs_new_protected:Npn \@@_rotate_box_of_block:
7265 {
7266     \box_grotate:cn
7267     { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7268     { 90 }
7269     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7270     {
7271         \vbox_gset_top:cn
7272         { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7273         {
7274             \skip_vertical:n { 0.8 ex }
7275             \box_use:c
7276             { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7277         }
7278     }
7279     \bool_if:NT \g_@@_rotate_c_bool
7280     {
7281         \hbox_gset:cn
7282         { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7283         {
7284             \c_math_toggle_token
7285             \vcenter
7286             {
7287                 \box_use:c
7288                 { g_@@_block_box_int \int_use:N \g_@@_block_box_int_box }
7289             }
7290             \c_math_toggle_token
7291         }
7292     }
7293 }

```

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.

```

7294 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7295 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7296 {
7297   \seq_gput_right:Ne \g_@@_blocks_seq
7298   {
7299     \l_tmpa_tl
7300     { \exp_not:n { #3 } }
7301     {
7302       \bool_if:NTF \l_@@_tabular_bool
7303       {
7304         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7305     \@@_reset_arraystretch:
7306     \exp_not:n
7307     {
7308       \dim_zero:N \extrarowheight
7309       #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).

```

7310       \bool_if:NT \c_@@_testphase_table_bool
7311       { \tag_stop:n { table } }
7312       \use:e
7313       {
7314         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7315         { @ { } \l_@@_hpos_block_str @ { } }
7316       }
7317       #5
7318       \end { tabular }
7319     }
7320   \group_end:
7321 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7322   {
7323     \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7324     \@@_reset_arraystretch:
7325     \exp_not:n
7326     {
7327       \dim_zero:N \extrarowheight
7328       #4
7329       \c_math_toggle_token
7330       \use:e
7331       {
7332         \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7333         { @ { } \l_@@_hpos_block_str @ { } }
7334       }
7335       #5
7336       \end { array }
7337       \c_math_toggle_token
7338     }

```

```

7339         \group_end:
7340     }
7341 }
7342 }
7343 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7344 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7345 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7346 {
7347     \seq_gput_right:Ne \g_@@_blocks_seq
7348     {
7349         \l_tmpa_tl
7350         { \exp_not:n { #3 } }
7351         {
7352             \group_begin:
7353             \exp_not:n { #4 #5 }
7354             \group_end:
7355         }
7356     }
7357 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7358 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7359 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7360 {
7361     \seq_gput_right:Ne \g_@@_blocks_seq
7362     {
7363         \l_tmpa_tl
7364         { \exp_not:n { #3 } }
7365         { \exp_not:n { #4 #5 } }
7366     }
7367 }

```

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).

```

7368 \keys_define:nn { nicematrix / Block / SecondPass }
7369 {
7370     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7371     ampersand-in-blocks .default:n = true ,
7372     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7373     tikz .code:n =
7374         \IfPackageLoadedTF { tikz }
7375         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7376         { \@@_error:n { tikz-key-without-tikz } } ,
7377     tikz .value_required:n = true ,
7378     fill .code:n =
7379         \tl_set_rescan:Nnn
7380         \l_@@_fill_tl
7381         { \char_set_catcode_other:N ! }
7382         { #1 } ,
7383     fill .value_required:n = true ,
7384     opacity .tl_set:N = \l_@@_opacity_tl ,
7385     opacity .value_required:n = true ,
7386     draw .code:n =
7387         \tl_set_rescan:Nnn
7388         \l_@@_draw_tl
7389         { \char_set_catcode_other:N ! }

```

```

7390     { #1 } ,
7391 draw .default:n = default ,
7392 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7393 rounded-corners .default:n = 4 pt ,
7394 color .code:n =
7395   \@@_color:n { #1 }
7396   \tl_set_rescan:Nnn
7397     \l_@@_draw_tl
7398     { \char_set_catcode_other:N ! }
7399     { #1 } ,
7400 borders .clist_set:N = \l_@@_borders_clist ,
7401 borders .value_required:n = true ,
7402 hvlines .meta:n = { vlines , hlines } ,
7403 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7404 vlines .default:n = true ,
7405 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7406 hlines .default:n = true ,
7407 line-width .dim_set:N = \l_@@_line_width_dim ,
7408 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7409 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7410             \bool_set_true:N \l_@@_p_block_bool ,
7411 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7412 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7413 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7414 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7415             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7416 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7417             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7418 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7419             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7420 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7421 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7422 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7423 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7424 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7425 m .value_forbidden:n = true ,
7426 v-center .meta:n = m ,
7427 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7428 p .value_forbidden:n = true ,
7429 name .tl_set:N = \l_@@_block_name_str ,
7430 name .value_required:n = true ,
7431 name .initial:n = ,
7432 respect-arraystretch .code:n =
7433   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7434 respect-arraystretch .value_forbidden:n = true ,
7435 transparent .bool_set:N = \l_@@_transparent_bool ,
7436 transparent .default:n = true ,
7437 transparent .initial:n = false ,
7438 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7439 }

```

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.

```

7440 \cs_new_protected:Npn \@@_draw_blocks:
7441 {
7442   \bool_if:NTF \c_@@_tagging_array_bool
7443     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7444     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7445   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnn ##1 }
7446 }

```

```

7447 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7448 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7449 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7450 \int_zero_new:N \l_@@_last_row_int
7451 \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

7452 \int_compare:nNnTF { #3 } > { 99 }
7453 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7454 { \int_set:Nn \l_@@_last_row_int { #3 } }
7455 \int_compare:nNnTF { #4 } > { 99 }
7456 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7457 { \int_set:Nn \l_@@_last_col_int { #4 } }
7458 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7459 {
7460 \bool_lazy_and:nnTF
7461 \l_@@_preamble_bool
7462 {
7463 \int_compare_p:n
7464 { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7465 }
7466 {
7467 \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7468 \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7469 \@@_msg_redirect_name:nn { columns-not-used } { none }
7470 }
7471 { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7472 }
7473 {
7474 \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7475 { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7476 {
7477 \@@_Block_v:nneenn
7478 { #1 }
7479 { #2 }
7480 { \int_use:N \l_@@_last_row_int }
7481 { \int_use:N \l_@@_last_col_int }
7482 { #5 }
7483 { #6 }
7484 }
7485 }
7486 }

```

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

```

7487 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7488 {

```

The group is for the keys.

```

7489 \group_begin:
7490 \int_compare:nNnT { #1 } = { #3 }
7491 { \str_set:Nn \l_@@_vpos_block_str { t } }
7492 \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`.

```

7493   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7494   \bool_lazy_and:nnT
7495     \l_@@_vlines_block_bool
7496     { ! \l_@@_ampersand_bool }
7497     {
7498       \tl_gput_right:Ne \g_nicematrix_code_after_tl
7499       {
7500         \@@_vlines_block:nnn
7501         { \exp_not:n { #5 } }
7502         { #1 - #2 }
7503         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7504       }
7505     }
7506   \bool_if:NT \l_@@_hlines_block_bool
7507     {
7508       \tl_gput_right:Ne \g_nicematrix_code_after_tl
7509       {
7510         \@@_hlines_block:nnn
7511         { \exp_not:n { #5 } }
7512         { #1 - #2 }
7513         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7514       }
7515     }
7516   \bool_if:NF \l_@@_transparent_bool
7517     {
7518     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7519     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7520       \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7521       { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7522     }
7523   }
7524   \tl_if_empty:NF \l_@@_draw_tl
7525     {
7526     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7527     { \@@_error:n { hlines-with-color } }
7528     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7529     {
7530     \@@_stroke_block:nnn
7531     { \exp_not:n { #5 } }
7532     { #1 - #2 }
7533     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7534     }
7535     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7536     { { #1 } { #2 } { #3 } { #4 } }
7537     }
7538   \clist_if_empty:NF \l_@@_borders_clist
7539     {
7540     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7541     {
7542     \@@_stroke_borders_block:nnn
7543     { \exp_not:n { #5 } }
7544     { #1 - #2 }
7545     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7546     }
7547     }

```

#5 are the options


```

7548 \tl_if_empty:NF \l_@@_fill_tl
7549 {
7550   \tl_if_empty:NF \l_@@_opacity_tl
7551   {
7552     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7553       {
7554         \tl_set:Ne \l_@@_fill_tl
7555         {
7556           [ opacity = \l_@@_opacity_tl ,
7557             \tl_tail:o \l_@@_fill_tl
7558         ]
7559       }
7560     {
7561       \tl_set:Ne \l_@@_fill_tl
7562       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7563     }
7564   }
7565   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7566   {
7567     \exp_not:N \roundedrectanglecolor
7568     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7569       { \l_@@_fill_tl }
7570       { { \l_@@_fill_tl } }
7571       { #1 - #2 }
7572       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7573       { \dim_use:N \l_@@_rounded_corners_dim }
7574     }
7575   }
7576   \seq_if_empty:NF \l_@@_tikz_seq
7577   {
7578     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7579     {
7580       \@@_block_tikz:nnnnn
7581       { \seq_use:Nn \l_@@_tikz_seq { , } }
7582       { #1 }
7583       { #2 }
7584       { \int_use:N \l_@@_last_row_int }
7585       { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7586     }
7587   }

```

```

7588   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7589   {
7590     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7591     {
7592       \@@_actually_diagbox:nnnnnn
7593       { #1 }
7594       { #2 }
7595       { \int_use:N \l_@@_last_row_int }
7596       { \int_use:N \l_@@_last_col_int }
7597       { \exp_not:n { ##1 } }
7598       { \exp_not:n { ##2 } }
7599     }
7600   }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7601 \pgfpicture
7602 \pgfrememberpicturepositiononpagetrue
7603 \pgf@relevantforpicturesizefalse
7604 \@@_qpoint:n { row - #1 }
7605 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7606 \@@_qpoint:n { col - #2 }
7607 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7608 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7609 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7610 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7611 \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.

```

7612 \@@_pgf_rect_node:nnnnn
7613 { \@@_env: - #1 - #2 - block }
7614 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7615 \str_if_empty:NF \l_@@_block_name_str
7616 {
7617 \pgfnodealias
7618 { \@@_env: - \l_@@_block_name_str }
7619 { \@@_env: - #1 - #2 - block }
7620 \str_if_empty:NF \l_@@_name_str
7621 {
7622 \pgfnodealias
7623 { \l_@@_name_str - \l_@@_block_name_str }
7624 { \@@_env: - #1 - #2 - block }
7625 }
7626 }

```

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.

```

7627 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7628 {
7629 \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.

```

7630 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7631 {

```

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.

```

7632 \cs_if_exist:cT
7633 { pgf @ sh @ ns @ \@@_env: - #1 - #2 }

```

```

7634     {
7635     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7636     {
7637     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7638     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7639     }
7640     }
7641 }

```

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.

```

7642 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7643 {
7644 \@@_qpoint:n { col - #2 }
7645 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7646 }
7647 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7648 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7649 {
7650 \cs_if_exist:cT
7651 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7652 {
7653 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7654 {
7655 \pgfpointanchor
7656 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7657 { east }
7658 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7659 }
7660 }
7661 }
7662 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7663 {
7664 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7665 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7666 }
7667 \@@_pgf_rect_node:nnnnn
7668 { \@@_env: - #1 - #2 - block - short }
7669 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7670 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

7671 \bool_if:NT \l_@@_medium_nodes_bool
7672 {
7673 \@@_pgf_rect_node:nnn
7674 { \@@_env: - #1 - #2 - block - medium }
7675 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7676 {
7677 \pgfpointanchor
7678 { \@@_env:
7679 - \int_use:N \l_@@_last_row_int
7680 - \int_use:N \l_@@_last_col_int - medium
7681 }
7682 { south~east }
7683 }
7684 }
7685 \endpgfpicture

7686 \bool_if:NTF \l_@@_ampersand_bool
7687 {
7688 \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7689 \int_zero_new:N \l_@@_split_int

```

```

7690 \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7691 \pgfpicture
7692 \pgfrememberpicturepositiononpagetrue
7693 \pgf@relevantforpicturesizefalse
7694 \@@_qpoint:n { row - #1 }
7695 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7696 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7697 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7698 \@@_qpoint:n { col - #2 }
7699 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7700 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7701 \dim_set:Nn \l_tmpb_dim
7702 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7703 \bool_lazy_or:nnF
7704 \l_@@_vlines_block_bool
7705 { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7706 {
7707 \int_step_inline:nn { \l_@@_split_int - 1 }
7708 {
7709 \pgfpathmoveto
7710 {
7711 \pgfpoint
7712 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7713 \l_@@_tmpc_dim
7714 }
7715 \pgfpathlineto
7716 {
7717 \pgfpoint
7718 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7719 \l_@@_tmpd_dim
7720 }
7721 \CT@arc@
7722 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7723 \pgfsetrectcap
7724 \pgfusepathqstroke
7725 }
7726 }
7727 \@@_qpoint:n { row - #1 - base }
7728 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7729 \int_step_inline:nn \l_@@_split_int
7730 {
7731 \group_begin:
7732 \dim_set:Nn \col@sep
7733 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7734 \pgftransformshift
7735 {
7736 \pgfpoint
7737 {
7738 \str_case:on \l_@@_hpos_block_str
7739 {
7740 l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep }
7741 c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7742 r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7743 }
7744 }
7745 { \l_@@_tmpc_dim }
7746 }
7747 \pgfset
7748 {
7749 inner~xsep = \c_zero_dim ,
7750 inner~ysep = \c_zero_dim
7751 }
7752 \pgfnode

```

```

7753     { rectangle }
7754     {
7755         \str_case:on \l_@@_hpos_block_str
7756         {
7757             c { base }
7758             l { base-west }
7759             r { base-east }
7760         }
7761     }
7762     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7763     \group_end:
7764 }
7765 \endpgfpicture
7766 }

```

Now the case where there is no ampersand & in the content of the block.

```

7767 {
7768     \bool_if:NTF \l_@@_p_block_bool
7769     {

```

When the final user has used the key p, we have to compute the width.

```

7770         \pgfpicture
7771         \pgfrememberpicturepositiononpagetrue
7772         \pgf@relevantforpicturesizefalse
7773         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7774         {
7775             \@@_qpoint:n { col - #2 }
7776             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7777             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7778         }
7779         {
7780             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7781             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7782             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7783         }
7784         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7785     \endpgfpicture
7786     \hbox_set:Nn \l_@@_cell_box
7787     {
7788         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7789         { \g_tmpb_dim }
7790         \str_case:on \l_@@_hpos_block_str
7791         { c \centering r \raggedleft l \raggedright j { } }
7792         #6
7793         \end { minipage }
7794     }
7795 }
7796 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7797 \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.

```

7798     \pgfpicture
7799     \pgfrememberpicturepositiononpagetrue
7800     \pgf@relevantforpicturesizefalse
7801     \bool_lazy_any:nTF
7802     {
7803         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7804         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7805         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7806         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7807     }
7808     {

```

If we are in the first column, we must put the block as if it was with the key r.

```
7809         \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.

```
7810         \bool_if:nT \g_@@_last_col_found_bool
7811         {
7812             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7813             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7814         }
```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```
7815         \tl_set:Ne \l_tmpa_tl
7816         {
7817             \str_case:on \l_@@_vpos_block_str
7818             {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
7819             { } { % added 2024-06-29
7820                 \str_case:on \l_@@_hpos_block_str
7821                 {
7822                     c { center }
7823                     l { west }
7824                     r { east }
7825                     j { center }
7826                 }
7827             }
7828         c {
7829             \str_case:on \l_@@_hpos_block_str
7830             {
7831                 c { center }
7832                 l { west }
7833                 r { east }
7834                 j { center }
7835             }
7836         }
7837     }
7838     T {
7839         \str_case:on \l_@@_hpos_block_str
7840         {
7841             c { north }
7842             l { north-west }
7843             r { north-east }
7844             j { north }
7845         }
7846     }
7847 }
7848 B {
7849     \str_case:on \l_@@_hpos_block_str
7850     {
7851         c { south }
7852         l { south-west }
7853         r { south-east }
7854         j { south }
7855     }
7856 }
7857 }
7858 }
7859 }
7860 \pgftransformshift
7861 {
7862     \pgfpointanchor
7863     {
7864         \@@_env: - #1 - #2 - block
```

```

7865         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7866     }
7867     { \l_tmpa_tl }
7868 }
7869 \pgfset
7870 {
7871     inner~xsep = \c_zero_dim ,
7872     inner~ysep = \c_zero_dim
7873 }
7874 \pgfnode
7875 { rectangle }
7876 { \l_tmpa_tl }
7877 { \box_use_drop:N \l_@@_cell_box } { } { }
7878 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7879 {
7880     \pgfextracty \l_tmpa_dim
7881     {
7882         \@@_qpoint:n
7883         {
7884             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7885             - base
7886         }
7887     }
7888     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

7889     \pgfpointanchor
7890     {
7891         \@@_env: - #1 - #2 - block
7892         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7893     }
7894     {
7895         \str_case:on \l_@@_hpos_block_str
7896         {
7897             c { center }
7898             l { west }
7899             r { east }
7900             j { center }
7901         }
7902     }

```

We put the label of the block which has been composed in $\l_@@_cell_box$.

```

7903     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7904     \pgfset { inner~sep = \c_zero_dim }
7905     \pgfnode
7906     { rectangle }
7907     {
7908         \str_case:on \l_@@_hpos_block_str
7909         {
7910             c { base }
7911             l { base~west }
7912             r { base~east }
7913             j { base }
7914         }
7915     }
7916     { \box_use_drop:N \l_@@_cell_box } { } { }
7917 }
7918 \endpgfpicture
7919 }
7920 \group_end:
7921 }

```

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).

```

7922 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7923 {
7924   \group_begin:
7925   \tl_clear:N \l_@@_draw_tl
7926   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7927   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
7928   \pgfpicture
7929   \pgfrememberpicturepositiononpagetrue
7930   \pgf@relevantforpicturesizefalse
7931   \tl_if_empty:NF \l_@@_draw_tl
7932   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7933     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7934     { \CT@arc@ }
7935     { \@@_color:o \l_@@_draw_tl }
7936   }
7937   \pgfsetcornersarced
7938   {
7939     \pgfpoint
7940     { \l_@@_rounded_corners_dim }
7941     { \l_@@_rounded_corners_dim }
7942   }
7943   \@@_cut_on_hyphen:w #2 \q_stop
7944   \int_compare:nNnF \l_tmpa_tl > \c@iRow
7945   {
7946     \int_compare:nNnF \l_tmpb_tl > \c@jCol
7947     {
7948       \@@_qpoint:n { row - \l_tmpa_tl }
7949       \dim_set_eq:NN \l_tmpb_dim \pgf@y
7950       \@@_qpoint:n { col - \l_tmpb_tl }
7951       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7952       \@@_cut_on_hyphen:w #3 \q_stop
7953       \int_compare:nNnT \l_tmpa_tl > \c@iRow
7954       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7955       \int_compare:nNnT \l_tmpb_tl > \c@jCol
7956       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7957       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7958       \dim_set_eq:NN \l_tmpa_dim \pgf@y
7959       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7960       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7961       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7962       \pgfpathrectanglecorners
7963       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7964       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7965       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7966       { \pgfusepathqstroke }
7967       { \pgfusepath { stroke } }
7968     }
7969   }
7970   \endpgfpicture
7971   \group_end:
7972 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7973 \keys_define:nn { nicematrix / BlockStroke }
7974 {
7975   color .tl_set:N = \l_@@_draw_tl ,
7976   draw .code:n =
7977     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,

```



```

7978   draw .default:n = default ,
7979   line-width .dim_set:N = \l_@@_line_width_dim ,
7980   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7981   rounded-corners .default:n = 4 pt
7982 }

```

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).

```

7983 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7984 {
7985   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7986   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
7987   \@@_cut_on_hyphen:w #2 \q_stop
7988   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7989   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7990   \@@_cut_on_hyphen:w #3 \q_stop
7991   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7992   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7993   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7994   {
7995     \use:e
7996     {
7997       \@@_vline:n
7998       {
7999         position = ##1 ,
8000         start = \l_@@_tmpc_tl ,
8001         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8002         total-width = \dim_use:N \l_@@_line_width_dim
8003       }
8004     }
8005   }
8006 }
8007 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8008 {
8009   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8010   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8011   \@@_cut_on_hyphen:w #2 \q_stop
8012   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8013   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8014   \@@_cut_on_hyphen:w #3 \q_stop
8015   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8016   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8017   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8018   {
8019     \use:e
8020     {
8021       \@@_hline:n
8022       {
8023         position = ##1 ,
8024         start = \l_@@_tmpd_tl ,
8025         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8026         total-width = \dim_use:N \l_@@_line_width_dim
8027       }
8028     }
8029   }
8030 }

```

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).

```

8031 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3

```

```

8032 {
8033   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8034   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8035   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8036     { \@@_error:n { borders~forbidden } }
8037     {
8038       \tl_clear_new:N \l_@@_borders_tikz_tl
8039       \keys_set:no
8040         { nicematrix / OnlyForTikzInBorders }
8041         \l_@@_borders_clist
8042         \@@_cut_on_hyphen:w #2 \q_stop
8043         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8044         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8045         \@@_cut_on_hyphen:w #3 \q_stop
8046         \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8047         \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8048         \@@_stroke_borders_block_i:
8049       }
8050   }
8051   \hook_gput_code:nnn { begindocument } { . }
8052   {
8053     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8054       {
8055         \c_@@_pgfortikzpicture_tl
8056         \@@_stroke_borders_block_ii:
8057         \c_@@_endpgfortikzpicture_tl
8058       }
8059   }
8060   \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8061     {
8062       \pgfrememberpicturepositiononpagetrue
8063       \pgf@relevantforpicturesizefalse
8064       \CT@arc@
8065       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8066       \clist_if_in:NnT \l_@@_borders_clist { right }
8067         { \@@_stroke_vertical:n \l_tmpb_tl }
8068       \clist_if_in:NnT \l_@@_borders_clist { left }
8069         { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8070       \clist_if_in:NnT \l_@@_borders_clist { bottom }
8071         { \@@_stroke_horizontal:n \l_tmpa_tl }
8072       \clist_if_in:NnT \l_@@_borders_clist { top }
8073         { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8074     }
8075   \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8076     {
8077       tikz .code:n =
8078         \cs_if_exist:NTF \tikzpicture
8079           { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8080           { \@@_error:n { tikz~in~borders~without~tikz } } ,
8081       tikz .value_required:n = true ,
8082       top .code:n = ,
8083       bottom .code:n = ,
8084       left .code:n = ,
8085       right .code:n = ,
8086       unknown .code:n = \@@_error:n { bad~border }
8087     }

```

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).

```

8088 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8089   {
8090     \@@_qpoint:n \l_@@_tmpc_tl

```

```

8091 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8092 \@@_qpoint:n \l_tmpa_tl
8093 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8094 \@@_qpoint:n { #1 }
8095 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8096 {
8097   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8098   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8099   \pgfusepathqstroke
8100 }
8101 {
8102   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8103     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8104 }
8105 }

```

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).

```

8106 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8107 {
8108   \@@_qpoint:n \l_@@_tmpd_tl
8109   \clist_if_in:NnTF \l_@@_borders_clist { left }
8110     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8111     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8112   \@@_qpoint:n \l_tmpb_tl
8113   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8114   \@@_qpoint:n { #1 }
8115   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8116   {
8117     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8118     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8119     \pgfusepathqstroke
8120   }
8121   {
8122     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8123       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8124   }
8125 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8126 \keys_define:nn { nicematrix / BlockBorders }
8127 {
8128   borders .clist_set:N = \l_@@_borders_clist ,
8129   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8130   rounded-corners .default:n = 4 pt ,
8131   line-width .dim_set:N = \l_@@_line_width_dim
8132 }

```

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.

```

8133 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8134 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8135 {
8136   \begin { tikzpicture }
8137   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```
8138 \clist_map_inline:nn { #1 }
8139 {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8140 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8141 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8142 (
8143 [
8144 xshift = \dim_use:N \l_@@_offset_dim ,
8145 yshift = - \dim_use:N \l_@@_offset_dim
8146 ]
8147 #2 -| #3
8148 )
8149 rectangle
8150 (
8151 [
8152 xshift = - \dim_use:N \l_@@_offset_dim ,
8153 yshift = \dim_use:N \l_@@_offset_dim
8154 ]
8155 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8156 ) ;
8157 }
8158 \end { tikzpicture }
8159 }
```

```
8160 \keys_define:nn { nicematrix / SpecialOffset }
8161 { 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.

```
8162 \cs_new_protected:Npn \@@_NullBlock:
8163 { \@@_collect_options:n { \@@_NullBlock_i: } }
8164 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8165 { }
```

27 How to draw the dotted lines transparently

```
8166 \cs_set_protected:Npn \@@_renew_matrix:
8167 {
8168 \RenewDocumentEnvironment { pmatrix } { }
8169 { \pNiceMatrix }
8170 { \endpNiceMatrix }
8171 \RenewDocumentEnvironment { vmatrix } { }
8172 { \vNiceMatrix }
8173 { \endvNiceMatrix }
8174 \RenewDocumentEnvironment { Vmatrix } { }
8175 { \VNiceMatrix }
8176 { \endVNiceMatrix }
8177 \RenewDocumentEnvironment { bmatrix } { }
8178 { \bNiceMatrix }
8179 { \endbNiceMatrix }
8180 \RenewDocumentEnvironment { Bmatrix } { }
8181 { \BNiceMatrix }
8182 { \endBNiceMatrix }
8183 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8184 \keys_define:nn { nicematrix / Auto }
8185 {
8186   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8187   columns-type .value_required:n = true ,
8188   l .meta:n = { columns-type = l } ,
8189   r .meta:n = { columns-type = r } ,
8190   c .meta:n = { columns-type = c } ,
8191   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8192   delimiters / color .value_required:n = true ,
8193   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8194   delimiters / max-width .default:n = true ,
8195   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8196   delimiters .value_required:n = true ,
8197   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8198   rounded-corners .default:n = 4 pt
8199 }

8200 \NewDocumentCommand \AutoNiceMatrixWithDelims
8201 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8202 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8203 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8204 {

```

The group is for the protection of the keys.

```

8205   \group_begin:
8206   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8207   \use:e
8208   {
8209     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8210     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8211     [ \exp_not:o \l_tmpa_tl ]
8212   }
8213   \int_if_zero:nT \l_@@_first_row_int
8214   {
8215     \int_if_zero:nT \l_@@_first_col_int { & }
8216     \prg_replicate:nn { #4 - 1 } { & }
8217     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8218   }
8219   \prg_replicate:nn { #3 }
8220   {
8221     \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`).

```

8222     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8223     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8224   }
8225   \int_compare:nNnT \l_@@_last_row_int > { -2 }
8226   {
8227     \int_if_zero:nT \l_@@_first_col_int { & }
8228     \prg_replicate:nn { #4 - 1 } { & }
8229     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8230   }
8231   \end { NiceArrayWithDelims }
8232   \group_end:
8233 }

8234 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8235 {

```

```

8236 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8237 {
8238   \bool_gset_true:N \g_@@_delims_bool
8239   \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8240   \AutoNiceMatrixWithDelims { #2 } { #3 }
8241 }
8242 }
8243 \@@_define_com:nnn p ( )
8244 \@@_define_com:nnn b [ ]
8245 \@@_define_com:nnn v | |
8246 \@@_define_com:nnn V \! \! \!
8247 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8248 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8249 {
8250   \group_begin:
8251   \bool_gset_false:N \g_@@_delims_bool
8252   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8253   \group_end:
8254 }

```

29 The redefinition of the command `\dotfill`

```

8255 \cs_set_eq:NN \@@_old_dotfill \dotfill
8256 \cs_new_protected:Npn \@@_dotfill:
8257 {

```

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}`).

```

8258   \@@_old_dotfill
8259   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8260 }

```

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`.

```

8261 \cs_new_protected:Npn \@@_dotfill_i:
8262 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8263 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8264 {
8265   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8266   {
8267     \@@_actually_diagbox:nnnnn
8268     { \int_use:N \c@iRow }
8269     { \int_use:N \c@jCol }
8270     { \int_use:N \c@iRow }
8271     { \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.

```
8272     { \g_@@_row_style_tl \exp_not:n { #1 } }
8273     { \g_@@_row_style_tl \exp_not:n { #2 } }
8274 }
```

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.

```
8275 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8276 {
8277   { \int_use:N \c@iRow }
8278   { \int_use:N \c@jCol }
8279   { \int_use:N \c@iRow }
8280   { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8281   { }
8282 }
8283 }
```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```
8284 \cs_new_protected:Npn \@@_actually_diagbox:nnnnn #1 #2 #3 #4 #5 #6
8285 {
8286   \pgfpicture
8287   \pgf@relevantforpicturesizefalse
8288   \pgfrememberpicturepositiononpagetrue
8289   \@@_qpoint:n { row - #1 }
8290   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8291   \@@_qpoint:n { col - #2 }
8292   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8293   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8294   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8295   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8296   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8297   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8298   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8299 }
```

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.

```
8300   \CT@arc@
8301   \pgfsetroundcap
8302   \pgfusepathqstroke
8303 }
8304 \pgfset { inner~sep = 1 pt }
8305 \pgfscope
8306 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8307 \pgfnode { rectangle } { south~west }
8308 {
8309   \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```
8310   \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8311   \end { minipage }
8312 }
8313 { }
8314 { }
```

```

8315 \endpgfscope
8316 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8317 \pgfnode { rectangle } { north~east }
8318 {
8319 \begin { minipage } { 20 cm }
8320 \raggedleft
8321 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8322 \end { minipage }
8323 }
8324 { }
8325 { }
8326 \endpgfpicture
8327 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 83.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8328 \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 `\`.

```

8329 \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`.

```

8330 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8331 {
8332 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8333 \@@_CodeAfter_iv:n
8334 }

```

We catch the argument of the command `\end` (in `#1`).

```

8335 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8336 {

```

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.

```

8337 \str_if_eq:eeTF \@currenenv { #1 }
8338 { \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`.

```

8339 {
8340 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8341 \@@_CodeAfter_ii:n
8342 }
8343 }

```


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.

```
8344 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8345   {
8346     \pgfpicture
8347     \pgfrememberpicturepositiononpagetrue
8348     \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.

```
8349     \@@_qpoint:n { row - 1 }
8350     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8351     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8352     \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).

```
8353     \bool_if:nTF { #3 }
8354       { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8355       { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8356     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8357       {
8358         \cs_if_exist:cT
8359           { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8360           {
8361             \pgfpointanchor
8362               { \@@_env: - ##1 - #2 }
8363               { \bool_if:nTF { #3 } { west } { east } }
8364             \dim_set:Nn \l_tmpa_dim
8365               { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8366           }
8367       }
```

Now we can put the delimiter with a node of PGF.

```
8368     \pgfset { inner~sep = \c_zero_dim }
8369     \dim_zero:N \nulldelimiterspace
8370     \pgftransformshift
8371       {
8372         \pgfpoint
8373           { \l_tmpa_dim }
8374           { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8375       }
8376     \pgfnode
8377       { rectangle }
8378       { \bool_if:nTF { #3 } { east } { west } }
8379       {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
8380     \nullfont
8381     \c_math_toggle_token
8382     \@@_color:o \l_@@_delimiters_color_tl
8383     \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

8384     \vcenter
8385     {
8386         \nullfont
8387         \hrule \@height
8388             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8389             \@depth \c_zero_dim
8390             \@width \c_zero_dim
8391     }
8392     \bool_if:nTF { #3 } { \right . } { \right #1 }
8393     \c_math_toggle_token
8394 }
8395 { }
8396 { }
8397 \endpgfpicture
8398 }

```

33 The command `\SubMatrix`

```

8399 \keys_define:nn { nicematrix / sub-matrix }
8400 {
8401     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8402     extra-height .value_required:n = true ,
8403     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8404     left-xshift .value_required:n = true ,
8405     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8406     right-xshift .value_required:n = true ,
8407     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8408     xshift .value_required:n = true ,
8409     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8410     delimiters / color .value_required:n = true ,
8411     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8412     slim .default:n = true ,
8413     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8414     hlines .default:n = all ,
8415     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8416     vlines .default:n = all ,
8417     hvlines .meta:n = { hlines, vlines } ,
8418     hvlines .value_forbidden:n = true
8419 }
8420 \keys_define:nn { nicematrix }
8421 {
8422     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8423     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8424     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8425     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8426 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8427 \keys_define:nn { nicematrix / SubMatrix }
8428 {
8429     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8430     delimiters / color .value_required:n = true ,
8431     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8432     hlines .default:n = all ,
8433     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8434     vlines .default:n = all ,
8435     hvlines .meta:n = { hlines, vlines } ,
8436     hvlines .value_forbidden:n = true ,
8437     name .code:n =

```

```

8438 \tl_if_empty:nTF { #1 }
8439 { \@@_error:n { Invalid-name } }
8440 {
8441   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8442   {
8443     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8444     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8445     {
8446       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8447       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8448     }
8449   }
8450   { \@@_error:n { Invalid-name } }
8451 } ,
8452 name .value_required:n = true ,
8453 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8454 rules .value_required:n = true ,
8455 code .tl_set:N = \l_@@_code_tl ,
8456 code .value_required:n = true ,
8457 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8458 }

8459 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8460 {
8461   \peek_remove_spaces:n
8462   {
8463     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8464     {
8465       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8466       [
8467         delimiters / color = \l_@@_delimiters_color_tl ,
8468         hlines = \l_@@_submatrix_hlines_clist ,
8469         vlines = \l_@@_submatrix_vlines_clist ,
8470         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8471         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8472         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8473         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8474         #5
8475       ]
8476     }
8477     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8478   }
8479 }

8480 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8481 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8482 { \@@_SubMatrix_in_code_before_i:n n n n #1 #2 }

8483 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:n n n n #1 #2 #3 #4
8484 {
8485   \seq_gput_right:Ne \g_@@_submatrix_seq
8486   {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8487     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8488     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8489     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8490     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8491   }
8492 }

```

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`.

```

8493 \hook_gput_code:nnn { begindocument } { . }
8494 {
8495   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8496   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8497   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8498     {
8499       \peek_remove_spaces:n
8500         {
8501           \@@_sub_matrix:nnnnnnn
8502             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8503         }
8504     }
8505 }

```

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).

```

8506 \NewDocumentCommand \@@_compute_i_j:nn
8507 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8508 { \@@_compute_i_j:nnnn #1 #2 }
8509 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8510 {
8511   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8512   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8513   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8514   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8515   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8516     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8517   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8518     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8519   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8520     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8521   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8522     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8523 }
8524 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8525 {
8526   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8527 \@@_compute_i_j:nn { #2 } { #3 }
8528 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8529   { \cs_set_nopar:Npn \arraystretch { 1 } }
8530 \bool_lazy_or:nnTF
8531   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8532   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8533   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8534   {
8535     \str_clear_new:N \l_@@_submatrix_name_str
8536     \keys_set:nn { nicematrix / SubMatrix } { #5 }

```

```

8537 \pgfpicture
8538 \pgfrememberpicturepositiononpagetrue
8539 \pgf@relevantforpicturesizefalse
8540 \pgfset { inner~sep = \c_zero_dim }
8541 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8542 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8543 \bool_if:NTF \l_@@_submatrix_slim_bool
8544 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8545 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8546 {
8547   \cs_if_exist:cT
8548   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8549   {
8550     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8551     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8552     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8553   }
8554   \cs_if_exist:cT
8555   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8556   {
8557     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8558     \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8559     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8560   }
8561 }
8562 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8563 { \@@_error:nn { Impossible~delimiter } { left } }
8564 {
8565   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8566   { \@@_error:nn { Impossible~delimiter } { right } }
8567   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8568 }
8569 \endpgfpicture
8570 }
8571 \group_end:
8572 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8573 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8574 {
8575   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8576   \dim_set:Nn \l_@@_y_initial_dim
8577   {
8578     \fp_to_dim:n
8579     {
8580       \pgf@y
8581       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8582     }
8583   }
8584   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8585   \dim_set:Nn \l_@@_y_final_dim
8586   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8587   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8588   {
8589     \cs_if_exist:cT
8590     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8591     {
8592       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8593       \dim_set:Nn \l_@@_y_initial_dim
8594       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8595     }

```

```

8596     \cs_if_exist:cT
8597     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8598     {
8599         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8600         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8601         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8602     }
8603 }
8604 \dim_set:Nn \l_tmpa_dim
8605 {
8606     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8607     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8608 }
8609 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8610     \group_begin:
8611     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8612     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8613     \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`.

```

8614     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8615     {
8616         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8617         {
8618             \int_compare:nNnT
8619             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8620             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8621             \@@_qpoint:n { col - ##1 }
8622             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8623             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8624             \pgfusepathqstroke
8625         }
8626     }
8627 }

```

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.

```

8628     \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8629     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8630     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8631     {
8632         \bool_lazy_and:nnTF
8633         { \int_compare_p:nNn { ##1 } > \c_zero_int }
8634         {
8635             \int_compare_p:nNn
8636             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8637         {
8638             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8639             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8640             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8641             \pgfusepathqstroke
8642         }
8643         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8644     }

```

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.

```

8645 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8646 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8647 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8648 {
8649   \bool_lazy_and:nnTF
8650   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8651   {
8652     \int_compare_p:nNn
8653     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8654   {
8655     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8656   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8657     \dim_set:Nn \l_tmpa_dim
8658     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8659     \str_case:nn { #1 }
8660     {
8661       ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8662       [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8663       \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8664       ]
8665     }
8666     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8666     \dim_set:Nn \l_tmpb_dim
8667     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8668     \str_case:nn { #2 }
8669     {
8670       ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8671       ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8672       \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8673     }
8674     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8675     \pgfusepathqstroke
8676     \group_end:
8677   }
8678   { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8679 }

```

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).

```

8680   \str_if_empty:NF \l_@@_submatrix_name_str
8681   {
8682     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8683     \l_@@_x_initial_dim \l_@@_y_initial_dim
8684     \l_@@_x_final_dim \l_@@_y_final_dim
8685   }
8686   \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`.

```

8687   \begin { pgfscope }
8688   \pgftransformshift
8689   {
8690     \pgfpoint
8691     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8692     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8693   }
8694   \str_if_empty:NNTF \l_@@_submatrix_name_str
8695   { \@@_node_left:nn #1 { } }

```

```

8696     { \l_@@_node_left:nn #1 { \l_@@_env: - \l_@@_submatrix_name_str - left } }
8697 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8698 \pgftransformshift
8699 {
8700   \pgfpoint
8701     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8702     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8703 }
8704 \str_if_empty:NTF \l_@@_submatrix_name_str
8705 { \l_@@_node_right:nmmm #2 { } { #3 } { #4 } }
8706 {
8707   \l_@@_node_right:nmmm #2
8708   { \l_@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8709 }
8710 \cs_set_eq:NN \pgfpointanchor \l_@@_pgfpointanchor:n
8711 \flag_clear_new:N \l_@@_code_flag
8712 \l_@@_code_tl
8713 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $row-i$, $col-j$ and $i|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8714 \cs_set_eq:NN \l_@@_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 `\l_@@_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.

```

8715 \cs_new_protected:Npn \l_@@_pgfpointanchor_i:nn #1
8716 {
8717   \use:e
8718   { \exp_not:N \l_@@_old_pgfpointanchor { \l_@@_pgfpointanchor_i:nn #1 } }
8719 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where "name_of_node" is the name of the Tikz node without the potential prefix and suffix. That's why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```

8720 \cs_new:Npn \l_@@_pgfpointanchor_i:nn #1 #2
8721 { #1 { \l_@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8722 \tl_const:Nn \c_@@_integers_alist_tl
8723 {
8724   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8725   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8726   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8727   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8728 }
8729 \cs_new:Npn \l_@@_pgfpointanchor_ii:w #1-#2\q_stop
8730 {

```


If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form $i-j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

8731 \tl_if_empty:nTF { #2 }
8732 {
8733   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8734   {
8735     \flag_raise:N \l_@@_code_flag
8736     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8737     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8738     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8739   }
8740   { #1 }
8741 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8742 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8743 }

```

There was an hyphen in the name of the node and that's why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

8744 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8745 {
8746   \str_case:nnF { #1 }
8747   {
8748     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8749     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8750   }

```

Now the case of a node of the form $i-j$.

```

8751 {
8752   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8753   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8754 }
8755 }

```

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`).

```

8756 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8757 {
8758   \pgfnode
8759   { rectangle }
8760   { east }
8761   {
8762     \nullfont
8763     \c_math_toggle_token
8764     \@@_color:o \l_@@_delimiters_color_tl
8765     \left #1
8766     \vcenter
8767     {
8768       \nullfont
8769       \hrule \@height \l_tmpa_dim
8770       \@depth \c_zero_dim
8771       \@width \c_zero_dim
8772     }
8773     \right .
8774     \c_math_toggle_token
8775   }
8776   { #2 }

```

```

8777     { }
8778   }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key name has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

8779 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8780 {
8781   \pgfnode
8782     { rectangle }
8783     { west }
8784     {
8785       \nullfont
8786       \c_math_toggle_token
8787       \colorlet { current-color } { . }
8788       \@@_color:o \l_@@_delimiters_color_tl
8789       \left .
8790       \vcenter
8791         {
8792           \nullfont
8793           \hrule \@height \l_tmpa_dim
8794             \@depth \c_zero_dim
8795             \@width \c_zero_dim
8796         }
8797       \right #1
8798       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8799       ^ { \color { current-color } \smash { #4 } }
8800       \c_math_toggle_token
8801     }
8802   { #2 }
8803   { }
8804 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8805 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8806 {
8807   \peek_remove_spaces:n
8808   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8809 }
8810 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8811 {
8812   \peek_remove_spaces:n
8813   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8814 }
8815 \keys_define:nn { nicematrix / Brace }
8816 {
8817   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8818   left-shorten .default:n = true ,
8819   left-shorten .value_forbidden:n = true ,
8820   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8821   right-shorten .default:n = true ,
8822   right-shorten .value_forbidden:n = true ,
8823   shorten .meta:n = { left-shorten , right-shorten } ,
8824   shorten .value_forbidden:n = true ,
8825   yshift .dim_set:N = \l_@@_brace_yshift_dim ,

```

```

8826     yshift .value_required:n = true ,
8827     yshift .initial:n = \c_zero_dim ,
8828     color .tl_set:N = \l_tmpa_tl ,
8829     color .value_required:n = true ,
8830     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8831 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to under or over.

```

8832 \cs_new_protected:Npn \@@_brace:nnnn #1 #2 #3 #4 #5
8833 {
8834   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8835   \@@_compute_i_j:nn { #1 } { #2 }
8836   \bool_lazy_or:nnTF
8837     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8838     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8839     {
8840       \str_if_eq:eeTF { #5 } { under }
8841       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8842       { \@@_error:nn { Construct-too-large } { \OverBrace } }
8843     }
8844   {
8845     \tl_clear:N \l_tmpa_tl
8846     \keys_set:nn { nicematrix / Brace } { #4 }
8847     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8848     \pgfpicture
8849     \pgfrememberpicturepositiononpagetrue
8850     \pgf@relevantforpicturesizefalse
8851     \bool_if:NT \l_@@_brace_left_shorten_bool
8852     {
8853       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8854       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8855       {
8856         \cs_if_exist:cT
8857           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8858           {
8859             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8860           }
8861         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8862           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8863       }
8864     }
8865   }
8866   \bool_lazy_or:nnT
8867     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8868     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8869     {
8870       \@@_qpoint:n { col - \l_@@_first_j_tl }
8871       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8872     }
8873   \bool_if:NT \l_@@_brace_right_shorten_bool
8874   {
8875     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8876     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8877     {
8878       \cs_if_exist:cT
8879         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8880         {
8881           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8882           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8883             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }

```

```

8884     }
8885     }
8886     }
8887     \bool_lazy_or:nnT
8888     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8889     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8890     {
8891       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8892       \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8893     }
8894     \pgfset { inner~sep = \c_zero_dim }
8895     \str_if_eq:eeTF { #5 } { under }
8896     { \@@_underbrace_i:n { #3 } }
8897     { \@@_overbrace_i:n { #3 } }
8898     \endpgfpicture
8899   }
8900 \group_end:
8901 }

```

The argument is the text to put above the brace.

```

8902 \cs_new_protected:Npn \@@_overbrace_i:n #1
8903 {
8904   \@@_qpoint:n { row - \l_@@_first_i_tl }
8905   \pgftransformshift
8906   {
8907     \pgfpoint
8908     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8909     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8910   }
8911   \pgfnode
8912   { rectangle }
8913   { south }
8914   {
8915     \vtop
8916     {
8917       \group_begin:
8918       \everycr { }
8919       \halign
8920       {
8921         \hfil ## \hfil \crcr
8922         \@@_math_toggle: #1 \@@_math_toggle: \cr
8923         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8924         \c_math_toggle_token
8925         \overbrace
8926         {
8927           \hbox_to_wd:nn
8928           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8929           { }
8930         }
8931         \c_math_toggle_token
8932       \cr
8933     }
8934     \group_end:
8935   }
8936 }
8937 { }
8938 { }
8939 }

```

The argument is the text to put under the brace.

```

8940 \cs_new_protected:Npn \@@_underbrace_i:n #1
8941 {
8942   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }

```

```

8943 \pgftransformshift
8944 {
8945   \pgfpoint
8946   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8947   { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8948 }
8949 \pgfnode
8950 { rectangle }
8951 { north }
8952 {
8953   \group_begin:
8954   \everycr { }
8955   \vbox
8956   {
8957     \halign
8958     {
8959       \hfil ## \hfil \crcr
8960       \c_math_toggle_token
8961       \underbrace
8962       {
8963         \hbox_to_wd:nn
8964         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8965         { }
8966       }
8967       \c_math_toggle_token
8968       \cr
8969       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8970       \c_math_toggle: #1 \c_math_toggle: \cr
8971     }
8972   }
8973   \group_end:
8974 }
8975 { }
8976 { }
8977 }

```

35 The command TikzEveryCell

```

8978 \bool_new:N \l_@@_not_empty_bool
8979 \bool_new:N \l_@@_empty_bool
8980
8981 \keys_define:nn { nicematrix / TikzEveryCell }
8982 {
8983   not-empty .code:n =
8984     \bool_lazy_or:nnTF
8985     \l_@@_in_code_after_bool
8986     \g_@@_recreate_cell_nodes_bool
8987     { \bool_set_true:N \l_@@_not_empty_bool }
8988     { \@@_error:n { detection~of~empty~cells } } } ,
8989   not-empty .value_forbidden:n = true ,
8990   empty .code:n =
8991     \bool_lazy_or:nnTF
8992     \l_@@_in_code_after_bool
8993     \g_@@_recreate_cell_nodes_bool
8994     { \bool_set_true:N \l_@@_empty_bool }
8995     { \@@_error:n { detection~of~empty~cells } } } ,
8996   empty .value_forbidden:n = true ,
8997   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8998 }

```

```

8999
9000
9001 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9002 {
9003   \IfPackageLoadedTF { tikz }
9004   {
9005     \group_begin:
9006     \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.

```

9007     \tl_set:Nn \l_tmpa_tl { { #2 } }
9008     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9009     { \@@_for_a_block:nnnnn #1 }
9010     \@@_all_the_cells:
9011     \group_end:
9012   }
9013   { \@@_error:n { TikzEveryCell~without~tikz } }
9014 }
9015
9016 \tl_new:N \@@_i_tl
9017 \tl_new:N \@@_j_tl
9018
9019
9020 \cs_new_protected:Nn \@@_all_the_cells:
9021 {
9022   \int_step_variable:nNn \c@iRow \@@_i_tl
9023   {
9024     \int_step_variable:nNn \c@jCol \@@_j_tl
9025     {
9026       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9027       {
9028         \clist_if_in:NcF \l_@@_corners_cells_clist
9029         { \@@_i_tl - \@@_j_tl }
9030         {
9031           \bool_set_false:N \l_tmpa_bool
9032           \cs_if_exist:cTF
9033           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9034           {
9035             \bool_if:NF \l_@@_empty_bool
9036             { \bool_set_true:N \l_tmpa_bool }
9037           }
9038           {
9039             \bool_if:NF \l_@@_not_empty_bool
9040             { \bool_set_true:N \l_tmpa_bool }
9041           }
9042           \bool_if:NT \l_tmpa_bool
9043           {
9044             \@@_block_tikz:nnnnn
9045             \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9046           }
9047         }
9048       }
9049     }
9050   }
9051 }
9052
9053 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9054 {
9055   \bool_if:NF \l_@@_empty_bool
9056   {
9057     \@@_block_tikz:nnnnn
9058     \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9059   }

```

```

9060 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9061 }
9062
9063 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9064 {
9065 \int_step_inline:nnn { #1 } { #3 }
9066 {
9067 \int_step_inline:nnn { #2 } { #4 }
9068 { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9069 }
9070 }

```

36 The command `\ShowCellNames`

```

9071 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9072 {
9073 \dim_gzero_new:N \g_@@_tmpc_dim
9074 \dim_gzero_new:N \g_@@_tmpd_dim
9075 \dim_gzero_new:N \g_@@_tmpe_dim
9076 \int_step_inline:nn \c@iRow
9077 {
9078 \begin { pgfpicture }
9079 \@@_qpoint:n { row - ##1 }
9080 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9081 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9082 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9083 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9084 \bool_if:NTF \l_@@_in_code_after_bool
9085 \end { pgfpicture }
9086 \int_step_inline:nn \c@jCol
9087 {
9088 \hbox_set:Nn \l_tmpa_box
9089 { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
9090 \begin { pgfpicture }
9091 \@@_qpoint:n { col - #####1 }
9092 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9093 \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9094 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9095 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9096 \endpgfpicture
9097 \end { pgfpicture }
9098 \fp_set:Nn \l_tmpa_fp
9099 {
9100 \fp_min:nn
9101 {
9102 \fp_min:nn
9103 {
9104 \dim_ratio:nn
9105 { \g_@@_tmpd_dim }
9106 { \box_wd:N \l_tmpa_box }
9107 }
9108 {
9109 \dim_ratio:nn
9110 { \g_tmpb_dim }
9111 { \box_ht_plus_dp:N \l_tmpa_box }
9112 }
9113 }
9114 { 1.0 }
9115 }
9116 \box_scale:Nnn \l_tmpa_box
9117 { \fp_use:N \l_tmpa_fp }
9118 { \fp_use:N \l_tmpa_fp }

```

```

9119     \pgfpicture
9120     \pgfrememberpicturepositiononpagetrue
9121     \pgf@relevantforpicturesizefalse
9122     \pgftransformshift
9123     {
9124         \pgfpoint
9125         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9126         { \dim_use:N \g_tmpa_dim }
9127     }
9128     \pgfnode
9129     { rectangle }
9130     { center }
9131     { \box_use:N \l_tmpa_box }
9132     { }
9133     { }
9134     \endpgfpicture
9135 }
9136 }
9137 }
9138 \NewDocumentCommand \@@_ShowCellNames { }
9139 {
9140     \bool_if:NT \l_@@_in_code_after_bool
9141     {
9142         \pgfpicture
9143         \pgfrememberpicturepositiononpagetrue
9144         \pgf@relevantforpicturesizefalse
9145         \pgfpathrectanglecorners
9146         { \@@_qpoint:n { 1 } }
9147         {
9148             \@@_qpoint:n
9149             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9150         }
9151         \pgfsetfillopacity { 0.75 }
9152         \pgfsetfillcolor { white }
9153         \pgfusepathqfill
9154         \endpgfpicture
9155     }
9156     \dim_gzero_new:N \g_@@_tmpc_dim
9157     \dim_gzero_new:N \g_@@_tmpd_dim
9158     \dim_gzero_new:N \g_@@_tmpe_dim
9159     \int_step_inline:nn \c@iRow
9160     {
9161         \bool_if:NTF \l_@@_in_code_after_bool
9162         {
9163             \pgfpicture
9164             \pgfrememberpicturepositiononpagetrue
9165             \pgf@relevantforpicturesizefalse
9166         }
9167         { \begin { pgfpicture } }
9168         \@@_qpoint:n { row - ##1 }
9169         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9170         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9171         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9172         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9173         \bool_if:NTF \l_@@_in_code_after_bool
9174         { \endpgfpicture }
9175         { \end { pgfpicture } }
9176         \int_step_inline:nn \c@jCol
9177         {
9178             \hbox_set:Nn \l_tmpa_box
9179             {
9180                 \normalfont \Large \sffamily \bfseries
9181                 \bool_if:NTF \l_@@_in_code_after_bool

```



```

9182         { \color { red } }
9183         { \color { red ! 50 } }
9184     ##1 - ####1
9185     }
9186     \bool_if:NTF \l_@@_in_code_after_bool
9187     {
9188         \pgfpicture
9189         \pgfrememberpicturepositiononpagetrue
9190         \pgf@relevantforpicturesizefalse
9191     }
9192     { \begin { pgfpicture } }
9193     @@_qpoint:n { col - ####1 }
9194     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9195     @@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9196     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9197     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9198     \bool_if:NTF \l_@@_in_code_after_bool
9199     { \endpgfpicture }
9200     { \end { pgfpicture } }
9201     \fp_set:Nn \l_tmpa_fp
9202     {
9203         \fp_min:nn
9204         {
9205             \fp_min:nn
9206             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9207             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9208         }
9209         { 1.0 }
9210     }
9211     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9212     \pgfpicture
9213     \pgfrememberpicturepositiononpagetrue
9214     \pgf@relevantforpicturesizefalse
9215     \pgftransformshift
9216     {
9217         \pgfpoint
9218         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9219         { \dim_use:N \g_tmpa_dim }
9220     }
9221     \pgfnode
9222     { rectangle }
9223     { center }
9224     { \box_use:N \l_tmpa_box }
9225     { }
9226     { }
9227     \endpgfpicture
9228     }
9229     }
9230 }

```

37 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

9231 \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.

```

9232 \bool_new:N \g_@@_footnote_bool
9233 \msg_new:nmmn { nicematrix } { Unknown~key~for~package }
9234 {
9235   The~key~'\l_keys_key_str'~is~unknown. \\
9236   That~key~will~be~ignored. \\
9237   For~a~list~of~the~available~keys,~type~H~<return>.
9238 }
9239 {
9240   The~available~keys~are~(in~alphabetic~order):~
9241   footnote,~
9242   footnotehyper,~
9243   messages~for~Overleaf,~
9244   renew~dots,~and~
9245   renew~matrix.
9246 }
9247 \keys_define:nn { nicematrix / Package }
9248 {
9249   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9250   renew~dots .value_forbidden:n = true ,
9251   renew~matrix .code:n = \@@_renew_matrix: ,
9252   renew~matrix .value_forbidden:n = true ,
9253   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9254   footnote .bool_set:N = \g_@@_footnote_bool ,
9255   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. You keep the following key only for compatibility but maybe we will delete it.

```

9256   no~test~for~array .code:n = \prg_do_nothing: ,
9257   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9258 }
9259 \ProcessKeysOptions { nicematrix / Package }

9260 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9261 {
9262   You~can't~use~the~option~'footnote'~because~the~package~
9263   footnotehyper~has~already~been~loaded.~
9264   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9265   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9266   of~the~package~footnotehyper.\\
9267   The~package~footnote~won't~be~loaded.
9268 }
9269 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9270 {
9271   You~can't~use~the~option~'footnotehyper'~because~the~package~
9272   footnote~has~already~been~loaded.~
9273   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9274   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9275   of~the~package~footnote.\\
9276   The~package~footnotehyper~won't~be~loaded.
9277 }

9278 \bool_if:NT \g_@@_footnote_bool
9279 {
9280   \IfClassLoadedTF { beamer }
9281     { \bool_set_false:N \g_@@_footnote_bool }
9282     {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9283     \IfPackageLoadedTF { footnotehyper }
9284     { \@@_error:n { footnote~with~footnotehyper~package } }
9285     { \usepackage { footnote } }
9286   }
9287 }
9288 \bool_if:NT \g_@@_footnotehyper_bool
9289 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9290   \IfClassLoadedTF { beamer }
9291   { \bool_set_false:N \g_@@_footnote_bool }
9292   {
9293     \IfPackageLoadedTF { footnote }
9294     { \@@_error:n { footnotehyper~with~footnote~package } }
9295     { \usepackage { footnotehyper } }
9296   }
9297   \bool_set_true:N \g_@@_footnote_bool
9298 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

38 About the package underscore

If the user loads the package underscore, it must be loaded *before* the package nicematrix. If it is loaded after, we raise an error.

```

9299 \bool_new:N \l_@@_underscore_loaded_bool
9300 \IfPackageLoadedT { underscore }
9301 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9302 \hook_gput_code:nnn { begindocument } { . }
9303 {
9304   \bool_if:NF \l_@@_underscore_loaded_bool
9305   {
9306     \IfPackageLoadedT { underscore }
9307     { \@@_error:n { underscore~after~nicematrix } }
9308   }
9309 }

```

39 Error messages of the package

```

9310 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9311 { \str_const:Nn \c_@@_available_keys_str { } }
9312 {
9313   \str_const:Nn \c_@@_available_keys_str
9314   { For~a~list~of~the~available~keys,~type~H~<return>. }
9315 }
9316 \seq_new:N \g_@@_types_of_matrix_seq
9317 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9318 {
9319   NiceMatrix ,
9320   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9321 }
9322 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq

```

```
9323 { \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`.

```
9324 \cs_new_protected:Npn \@@_error_too_much_cols:
9325 {
9326   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9327   { \@@_fatal:n { too-much-cols-for-array } }
9328   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9329   { \@@_fatal:n { too-much-cols-for-matrix } }
9330   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9331   { \@@_fatal:n { too-much-cols-for-matrix } }
9332   \bool_if:NF \l_@@_last_col_without_value_bool
9333   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9334 }
```

The following command must *not* be protected since it's used in an error message.

```
9335 \cs_new:Npn \@@_message_hdotsfor:
9336 {
9337   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9338   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9339 }
9340 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9341 {
9342   Incompatible~options.\\
9343   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9344   The~output~will~not~be~reliable.
9345 }
9346 \@@_msg_new:nn { negative~weight }
9347 {
9348   Negative~weight.\\
9349   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9350   the~value~'\int_use:N \l_@@_weight_int'.\\
9351   The~absolute~value~will~be~used.
9352 }
9353 \@@_msg_new:nn { last~col~not~used }
9354 {
9355   Column~not~used.\\
9356   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9357   in~your~\@@_full_name_env:..~However,~you~can~go~on.
9358 }
9359 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9360 {
9361   Too-much-columns.\\
9362   In~the~row~\int_eval:n { \c@iRow },~
9363   you~try~to~use~more~columns~
9364   than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9365   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9366   (plus~the~exterior~columns).~This~error~is~fatal.
9367 }
9368 \@@_msg_new:nn { too-much-cols-for-matrix }
9369 {
9370   Too-much-columns.\\
9371   In~the~row~\int_eval:n { \c@iRow },~
9372   you~try~to~use~more~columns~than~allowed~by~your~
9373   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9374   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9375   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9376   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
```

```

9377 \token_to_str:N \setcounter\ to-change-that-value).~
9378 This-error-is-fatal.
9379 }

9380 \@@_msg_new:nn { too-much-cols-for-array }
9381 {
9382   Too-much-columns.\\
9383   In-the-row~\int_eval:n { \c@iRow },~
9384   ~you-try-to-use-more-columns-than-allowed-by-your~
9385   \@@_full_name_env:\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9386   \int_use:N \g_@@_static_num_of_col_int\
9387   ~(plus-the-potential-exterior-ones).~
9388   This-error-is-fatal.
9389 }

9390 \@@_msg_new:nn { columns-not-used }
9391 {
9392   Columns-not-used.\\
9393   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9394   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9395   The-columns-you-did-not-used-won't-be-created.\\
9396   You-won't-have-similar-error-message-till-the-end-of-the-document.
9397 }

9398 \@@_msg_new:nn { empty-preamble }
9399 {
9400   Empty-preamble.\\
9401   The-preamble-of-your~\@@_full_name_env:\ is-empty.\\
9402   This-error-is-fatal.
9403 }

9404 \@@_msg_new:nn { in-first-col }
9405 {
9406   Erroneous-use.\\
9407   You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9408   That-command-will-be-ignored.
9409 }

9410 \@@_msg_new:nn { in-last-col }
9411 {
9412   Erroneous-use.\\
9413   You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9414   That-command-will-be-ignored.
9415 }

9416 \@@_msg_new:nn { in-first-row }
9417 {
9418   Erroneous-use.\\
9419   You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9420   That-command-will-be-ignored.
9421 }

9422 \@@_msg_new:nn { in-last-row }
9423 {
9424   You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9425   That-command-will-be-ignored.
9426 }

9427 \@@_msg_new:nn { caption-outside-float }
9428 {
9429   Key-caption-forbidden.\\
9430   You-can't-use-the-key~'caption'~because-you-are-not-in-a-floating~
9431   environment.~This-key-will-be-ignored.
9432 }

9433 \@@_msg_new:nn { short-caption-without-caption }
9434 {
9435   You-should-not-use-the-key~'short-caption'~without~'caption'.~

```

```

9436     However, ~your~ 'short-caption' ~will~be~used~as~ 'caption'.
9437 }
9438 \@@_msg_new:nn { double~closing~delimiter }
9439 {
9440     Double~delimiter.\\
9441     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9442     delimiter.~This~delimiter~will~be~ignored.
9443 }
9444 \@@_msg_new:nn { delimiter~after~opening }
9445 {
9446     Double~delimiter.\\
9447     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9448     delimiter.~That~delimiter~will~be~ignored.
9449 }
9450 \@@_msg_new:nn { bad~option~for~line~style }
9451 {
9452     Bad~line~style.\\
9453     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9454     is~'standard'.~That~key~will~be~ignored.
9455 }
9456 \@@_msg_new:nn { Identical~notes~in~caption }
9457 {
9458     Identical~tabular~notes.\\
9459     You~can't~put~several~notes~with~the~same~content~in~
9460     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9461     If~you~go~on,~the~output~will~probably~be~erroneous.
9462 }
9463 \@@_msg_new:nn { tabularnote~below~the~tabular }
9464 {
9465     \token_to_str:N \tabularnote\ forbidden\\
9466     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9467     of~your~tabular~because~the~caption~will~be~composed~below~
9468     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9469     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9470     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9471     no~similar~error~will~raised~in~this~document.
9472 }
9473 \@@_msg_new:nn { Unknown~key~for~rules }
9474 {
9475     Unknown~key.\\
9476     There~is~only~two~keys~available~here:~width~and~color.\\
9477     Your~key~'\l_keys_key_str'~will~be~ignored.
9478 }
9479 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9480 {
9481     Unknown~key.\\
9482     There~is~only~two~keys~available~here:~
9483     'empty'~and~'not~empty'.\\
9484     Your~key~'\l_keys_key_str'~will~be~ignored.
9485 }
9486 \@@_msg_new:nn { Unknown~key~for~rotate }
9487 {
9488     Unknown~key.\\
9489     The~only~key~available~here~is~'c'.\\
9490     Your~key~'\l_keys_key_str'~will~be~ignored.
9491 }
9492 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9493 {
9494     Unknown~key.\\
9495     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~

```

```

9496     It~you~go~on~,~you~will~probably~have~other~errors.  \\
9497     \c_@@_available_keys_str
9498   }
9499   {
9500     The~available~keys~are~(in~alphabetic~order):~
9501     ccommand,~
9502     color,~
9503     command,~
9504     dotted,~
9505     letter,~
9506     multiplicity,~
9507     sep-color,~
9508     tikz,~and~total~width.
9509   }

9510 \@@_msg_new:nnn { Unknown~key~for~xdots }
9511 {
9512   Unknown~key.\\
9513   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9514   \c_@@_available_keys_str
9515 }
9516 {
9517   The~available~keys~are~(in~alphabetic~order):~
9518   'color',~
9519   'horizontal-labels',~
9520   'inter',~
9521   'line-style',~
9522   'radius',~
9523   'shorten',~
9524   'shorten-end'~and~'shorten-start'.
9525 }

9526 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9527 {
9528   Unknown~key.\\
9529   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9530   (and~you~try~to~use~'\l_keys_key_str')\\
9531   That~key~will~be~ignored.
9532 }

9533 \@@_msg_new:nn { label~without~caption }
9534 {
9535   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9536   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9537 }

9538 \@@_msg_new:nn { W~warning }
9539 {
9540   Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9541   (row~\int_use:N \c@iRow).
9542 }

9543 \@@_msg_new:nn { Construct~too~large }
9544 {
9545   Construct~too~large.\\
9546   Your~command~\token_to_str:N #1
9547   can't~be~drawn~because~your~matrix~is~too~small.\\
9548   That~command~will~be~ignored.
9549 }

9550 \@@_msg_new:nn { underscore~after~nicematrix }
9551 {
9552   Problem~with~'underscore'.\\
9553   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9554   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9555   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{-times}}'.
9556 }

```

```

9557 \@@_msg_new:nn { ampersand-in-light-syntax }
9558 {
9559   Ampersand~forbidden.\
9560   You~can't~use~an~ampersand~(\token_to_str:N &)-to~separate~columns~because~
9561   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9562 }

9563 \@@_msg_new:nn { double-backslash-in-light-syntax }
9564 {
9565   Double~backslash~forbidden.\
9566   You~can't~use~\token_to_str:N
9567   \~to~separate~rows~because~the~key~'light-syntax'~
9568   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9569   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9570 }

9571 \@@_msg_new:nn { hlines-with-color }
9572 {
9573   Incompatible~keys.\
9574   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9575   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\
9576   However,~you~can~put~several~commands~\token_to_str:N \Block.\
9577   Your~key~will~be~discarded.
9578 }

9579 \@@_msg_new:nn { bad-value-for-baseline }
9580 {
9581   Bad~value~for~baseline.\
9582   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9583   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9584   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9585   the~form~'line-i'.\
9586   A~value~of~1~will~be~used.
9587 }

9588 \@@_msg_new:nn { detection-of-empty-cells }
9589 {
9590   Problem~with~'not-empty'\
9591   For~technical~reasons,~you~must~activate~
9592   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9593   in~order~to~use~the~key~'\l_keys_key_str'.\
9594   That~key~will~be~ignored.
9595 }

9596 \@@_msg_new:nn { siunitx-not-loaded }
9597 {
9598   siunitx~not~loaded\
9599   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\
9600   That~error~is~fatal.
9601 }

9602 \@@_msg_new:nn { Invalid-name }
9603 {
9604   Invalid~name.\
9605   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9606   \SubMatrix\ of~your~\@@_full_name_env:.\
9607   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\
9608   This~key~will~be~ignored.
9609 }

9610 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9611 {
9612   Wrong~line.\
9613   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9614   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9615   number~is~not~valid.~It~will~be~ignored.
9616 }

```



```

9617 \@@_msg_new:nn { Impossible-delimiter }
9618 {
9619   Impossible~delimiter.\\
9620   It's~impossible~to~draw~the~#1~delimiter~of~your~
9621   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9622   in~that~column.
9623   \bool_if:NT \l_@@_submatrix_slim_bool
9624   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9625   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9626 }
9627 \@@_msg_new:nnn { width-without-X-columns }
9628 {
9629   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9630   That~key~will~be~ignored.
9631 }
9632 {
9633   This~message~is~the~message~'width~without~X~columns'~
9634   of~the~module~'nicematrix'.~
9635   The~experimented~users~can~disable~that~message~with~
9636   \token_to_str:N \msg_redirect_name:nnn.\\
9637 }
9638
9639 \@@_msg_new:nn { key-multiplicity-with-dotted }
9640 {
9641   Incompatible~keys. \\
9642   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9643   in~a~'custom-line'.~They~are~incompatible. \\
9644   The~key~'multiplicity'~will~be~discarded.
9645 }
9646 \@@_msg_new:nn { empty-environment }
9647 {
9648   Empty~environment.\\
9649   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9650 }
9651 \@@_msg_new:nn { No~letter~and~no~command }
9652 {
9653   Erroneous~use.\\
9654   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9655   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9656   '~command'~(to~draw~horizontal~rules).\\
9657   However,~you~can~go~on.
9658 }
9659 \@@_msg_new:nn { Forbidden-letter }
9660 {
9661   Forbidden~letter.\\
9662   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9663   It~will~be~ignored.
9664 }
9665 \@@_msg_new:nn { Several~letters }
9666 {
9667   Wrong~name.\\
9668   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9669   have~used~'\l_@@_letter_str').\\
9670   It~will~be~ignored.
9671 }
9672 \@@_msg_new:nn { Delimiter~with~small }
9673 {
9674   Delimiter~forbidden.\\
9675   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9676   because~the~key~'small'~is~in~force.\\
9677   This~error~is~fatal.

```

```

9678 }
9679 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9680 {
9681   Unknown~cell.\\
9682   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9683   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9684   can't~be~executed~because~a~cell~doesn't~exist.\\
9685   This~command~\token_to_str:N \line\ will~be~ignored.
9686 }
9687 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9688 {
9689   Duplicate~name.\\
9690   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9691   in~this~\@@_full_name_env:.\\
9692   This~key~will~be~ignored.\\
9693   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9694   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9695 }
9696 {
9697   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9698   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9699 }
9700 \@@_msg_new:nn { r~or~l~with~preamble }
9701 {
9702   Erroneous~use.\\
9703   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~.~
9704   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9705   your~\@@_full_name_env:.\\
9706   This~key~will~be~ignored.
9707 }
9708 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9709 {
9710   Erroneous~use.\\
9711   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9712   the~array.~This~error~is~fatal.
9713 }
9714 \@@_msg_new:nn { bad~corner }
9715 {
9716   Bad~corner.\\
9717   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9718   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9719   This~specification~of~corner~will~be~ignored.
9720 }
9721 \@@_msg_new:nn { bad~border }
9722 {
9723   Bad~border.\\
9724   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9725   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9726   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9727   also~use~the~key~'tikz'
9728   \IfPackageLoadedF { tikz }
9729   {~if~you~load~the~LaTeX~package~'tikz'}).\\
9730   This~specification~of~border~will~be~ignored.
9731 }
9732 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9733 {
9734   TikZ~not~loaded.\\
9735   You~can't~use~\token_to_str:N \TikzEveryCell\
9736   because~you~have~not~loaded~tikz.~
9737   This~command~will~be~ignored.
9738 }

```

```

9739 \@@_msg_new:nn { tikz-key-without-tikz }
9740 {
9741   TikZ~not~loaded.\\
9742   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9743   \Block'~because~you~have~not~loaded~tikz.~
9744   This~key~will~be~ignored.
9745 }

9746 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
9747 {
9748   Erroneous~use.\\
9749   In~the~\@@_full_name_env:,~you~must~use~the~key~
9750   'last-col'~without~value.\\
9751   However,~you~can~go~on~for~this~time~
9752   (the~value~'\l_keys_value_tl'~will~be~ignored).
9753 }

9754 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9755 {
9756   Erroneous~use.\\
9757   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9758   'last-col'~without~value.\\
9759   However,~you~can~go~on~for~this~time~
9760   (the~value~'\l_keys_value_tl'~will~be~ignored).
9761 }

9762 \@@_msg_new:nn { Block-too-large-1 }
9763 {
9764   Block~too~large.\\
9765   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9766   too~small~for~that~block. \\
9767   This~block~and~maybe~others~will~be~ignored.
9768 }

9769 \@@_msg_new:nn { Block-too-large-2 }
9770 {
9771   Block~too~large.\\
9772   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9773   \g_@@_static_num_of_col_int\
9774   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9775   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9776   (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
9777   This~block~and~maybe~others~will~be~ignored.
9778 }

9779 \@@_msg_new:nn { unknown-column-type }
9780 {
9781   Bad~column~type.\\
9782   The~column~type~'#1'~in~your~\@@_full_name_env:\
9783   is~unknown. \\
9784   This~error~is~fatal.
9785 }

9786 \@@_msg_new:nn { unknown-column-type-S }
9787 {
9788   Bad~column~type.\\
9789   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9790   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9791   load~that~package. \\
9792   This~error~is~fatal.
9793 }

9794 \@@_msg_new:nn { tabularnote-forbidden }
9795 {
9796   Forbidden~command.\\
9797   You~can't~use~the~command~\token_to_str:N\tabularnote\
9798   ~here.~This~command~is~available~only~in~
9799   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~

```

```

9800     the~argument~of~a~command~\token_to_str:N \caption\ included-
9801     in~an~environment~{table}. \\
9802     This~command~will~be~ignored.
9803 }
9804 \@@_msg_new:nn { borders~forbidden }
9805 {
9806     Forbidden~key.\\
9807     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9808     because~the~option~'rounded~corners'~
9809     is~in~force~with~a~non~zero~value.\\
9810     This~key~will~be~ignored.
9811 }
9812 \@@_msg_new:nn { bottomrule~without~booktabs }
9813 {
9814     booktabs~not~loaded.\\
9815     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9816     loaded~'booktabs'.\\
9817     This~key~will~be~ignored.
9818 }
9819 \@@_msg_new:nn { enumitem~not~loaded }
9820 {
9821     enumitem~not~loaded.\\
9822     You~can't~use~the~command~\token_to_str:N\tabularnote\
9823     ~because~you~haven't~loaded~'enumitem'.\\
9824     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9825     ignored~in~the~document.
9826 }
9827 \@@_msg_new:nn { tikz~without~tikz }
9828 {
9829     Tikz~not~loaded.\\
9830     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9831     loaded.~If~you~go~on,~that~key~will~be~ignored.
9832 }
9833 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9834 {
9835     Tikz~not~loaded.\\
9836     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9837     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9838     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9839     use~that~custom~line.
9840 }
9841 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9842 {
9843     Tikz~not~loaded.\\
9844     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9845     command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9846     That~key~will~be~ignored.
9847 }
9848 \@@_msg_new:nn { without~color~inside }
9849 {
9850     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9851     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9852     outside~\token_to_str:N \CodeBefore,~you~
9853     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9854     You~can~go~on~but~you~may~need~more~compilations.
9855 }
9856 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9857 {
9858     Erroneous~use.\\
9859     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~

```

```

9860     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9861     The~key~'color'~will~be~discarded.
9862 }

9863 \@@_msg_new:nn { Wrong~last~row }
9864 {
9865     Wrong~number.\\
9866     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9867     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9868     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9869     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9870     without~value~(more~compilations~might~be~necessary).
9871 }

9872 \@@_msg_new:nn { Yet~in~env }
9873 {
9874     Nested~environments.\\
9875     Environments~of~nicematrix~can't~be~nested.\\
9876     This~error~is~fatal.
9877 }

9878 \@@_msg_new:nn { Outside~math~mode }
9879 {
9880     Outside~math~mode.\\
9881     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9882     (and~not~in~\token_to_str:N \vcenter).\\
9883     This~error~is~fatal.
9884 }

9885 \@@_msg_new:nn { One~letter~allowed }
9886 {
9887     Bad~name.\\
9888     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9889     It~will~be~ignored.
9890 }

9891 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9892 {
9893     Environment~{TabularNote}~forbidden.\\
9894     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9895     but~*before*~the~\token_to_str:N \CodeAfter.\\
9896     This~environment~{TabularNote}~will~be~ignored.
9897 }

9898 \@@_msg_new:nn { varwidth~not~loaded }
9899 {
9900     varwidth~not~loaded.\\
9901     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9902     loaded.\\
9903     Your~column~will~behave~like~'p'.
9904 }

9905 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9906 {
9907     Unkown~key.\\
9908     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9909     \c_@@_available_keys_str
9910 }
9911 {
9912     The~available~keys~are~(in~alphabetic~order):~
9913     color,~
9914     dotted,~
9915     multiplicity,~
9916     sep~color,~
9917     tikz,~and~total~width.
9918 }
9919

```

```

9920 \@@_msg_new:nnn { Unknown~key~for~Block }
9921 {
9922   Unknown~key.\\
9923   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9924   \Block.\\ It~will~be~ignored. \\
9925   \c_@@_available_keys_str
9926 }
9927 {
9928   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
9929   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
9930   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
9931   and~vlines.
9932 }
9933 \@@_msg_new:nnn { Unknown~key~for~Brace }
9934 {
9935   Unknown~key.\\
9936   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9937   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9938   It~will~be~ignored. \\
9939   \c_@@_available_keys_str
9940 }
9941 {
9942   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9943   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9944   right~shorten)~and~yshift.
9945 }
9946 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9947 {
9948   Unknown~key.\\
9949   The~key~'\l_keys_key_str'~is~unknown.\\
9950   It~will~be~ignored. \\
9951   \c_@@_available_keys_str
9952 }
9953 {
9954   The~available~keys~are~(in~alphabetic~order):~
9955   delimiters/color,~
9956   rules~(with~the~subkeys~'color'~and~'width'),~
9957   sub~matrix~(several~subkeys)~
9958   and~xdots~(several~subkeys).~
9959   The~latter~is~for~the~command~\token_to_str:N \line.
9960 }
9961 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9962 {
9963   Unknown~key.\\
9964   The~key~'\l_keys_key_str'~is~unknown.\\
9965   It~will~be~ignored. \\
9966   \c_@@_available_keys_str
9967 }
9968 {
9969   The~available~keys~are~(in~alphabetic~order):~
9970   create~cell~nodes,~
9971   delimiters/color~and~
9972   sub~matrix~(several~subkeys).
9973 }
9974 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9975 {
9976   Unknown~key.\\
9977   The~key~'\l_keys_key_str'~is~unknown.\\
9978   That~key~will~be~ignored. \\
9979   \c_@@_available_keys_str
9980 }
9981 {

```

```

9982 The~available~keys~are~(in~alphabetic~order):~
9983 'delimiters/color',~
9984 'extra-height',~
9985 'hlines',~
9986 'hvlines',~
9987 'left-xshift',~
9988 'name',~
9989 'right-xshift',~
9990 'rules'~(with~the~subkeys~'color'~and~'width'),~
9991 'slim',~
9992 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9993 and~'right-xshift').\\
9994 }
9995 \@@_msg_new:nnn { Unknown~key~for~notes }
9996 {
9997   Unknown~key.\\
9998   The~key~'\l_keys_key_str'~is~unknown.\\
9999   That~key~will~be~ignored. \\
10000   \c_@@_available_keys_str
10001 }
10002 {
10003   The~available~keys~are~(in~alphabetic~order):~
10004   bottomrule,~
10005   code-after,~
10006   code-before,~
10007   detect-duplicates,~
10008   enumitem-keys,~
10009   enumitem-keys-para,~
10010   para,~
10011   label-in-list,~
10012   label-in-tabular~and~
10013   style.
10014 }
10015 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10016 {
10017   Unknown~key.\\
10018   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10019   \token_to_str:N \RowStyle. \\
10020   That~key~will~be~ignored. \\
10021   \c_@@_available_keys_str
10022 }
10023 {
10024   The~available~keys~are~(in~alphabetic~order):~
10025   'bold',~
10026   'cell-space-top-limit',~
10027   'cell-space-bottom-limit',~
10028   'cell-space-limits',~
10029   'color',~
10030   'nb-rows'~and~
10031   'rowcolor'.
10032 }
10033 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10034 {
10035   Unknown~key.\\
10036   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10037   \token_to_str:N \NiceMatrixOptions. \\
10038   That~key~will~be~ignored. \\
10039   \c_@@_available_keys_str
10040 }
10041 {
10042   The~available~keys~are~(in~alphabetic~order):~
10043   &-in-blocks,~
10044   allow-duplicate-names,~

```

```

10045 ampersand-in-blocks,~
10046 caption-above,~
10047 cell-space-bottom-limit,~
10048 cell-space-limits,~
10049 cell-space-top-limit,~
10050 code-for-first-col,~
10051 code-for-first-row,~
10052 code-for-last-col,~
10053 code-for-last-row,~
10054 corners,~
10055 custom-key,~
10056 create-extra-nodes,~
10057 create-medium-nodes,~
10058 create-large-nodes,~
10059 custom-line,~
10060 delimiters~(several~subkeys),~
10061 end-of-row,~
10062 first-col,~
10063 first-row,~
10064 hlines,~
10065 hvlines,~
10066 hvlines-except-borders,~
10067 last-col,~
10068 last-row,~
10069 left-margin,~
10070 light-syntax,~
10071 light-syntax-expanded,~
10072 matrix/columns-type,~
10073 no-cell-nodes,~
10074 notes~(several~subkeys),~
10075 nullify-dots,~
10076 pgf-node-code,~
10077 renew-dots,~
10078 renew-matrix,~
10079 respect-arraystretch,~
10080 rounded-corners,~
10081 right-margin,~
10082 rules~(with~the~subkeys~'color'~and~'width'),~
10083 small,~
10084 sub-matrix~(several~subkeys),~
10085 vlines,~
10086 xdots~(several~subkeys).
10087 }

```

For ‘`{NiceArray}`’, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

10088 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10089 {
10090   Unknown~key.\\
10091   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10092   \{NiceArray\}. \\
10093   That~key~will~be~ignored. \\
10094   \c_@@_available_keys_str
10095 }
10096 {
10097   The~available~keys~are~(in~alphabetic~order):~
10098   &~in~blocks,~
10099   ampersand-in-blocks,~
10100   b,~
10101   baseline,~
10102   c,~
10103   cell-space-bottom-limit,~
10104   cell-space-limits,~
10105   cell-space-top-limit,~

```



```

10106 code-after,~
10107 code-for-first-col,~
10108 code-for-first-row,~
10109 code-for-last-col,~
10110 code-for-last-row,~
10111 color-inside,~
10112 columns-width,~
10113 corners,~
10114 create-extra-nodes,~
10115 create-medium-nodes,~
10116 create-large-nodes,~
10117 extra-left-margin,~
10118 extra-right-margin,~
10119 first-col,~
10120 first-row,~
10121 hlines,~
10122 hvlines,~
10123 hvlines-except-borders,~
10124 last-col,~
10125 last-row,~
10126 left-margin,~
10127 light-syntax,~
10128 light-syntax-expanded,~
10129 name,~
10130 no-cell-nodes,~
10131 nullify-dots,~
10132 pgf-node-code,~
10133 renew-dots,~
10134 respect-arraystretch,~
10135 right-margin,~
10136 rounded-corners,~
10137 rules~(with~the~subkeys~'color'~and~'width'),~
10138 small,~
10139 t,~
10140 vlines,~
10141 xdots/color,~
10142 xdots/shorten-start,~
10143 xdots/shorten-end,~
10144 xdots/shorten-and~
10145 xdots/line-style.
10146 }

```

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`).

```

10147 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10148 {
10149   Unknown~key.\\
10150   The~key~'\l_keys_key_str'~is~unknown~for~the~
10151   \@@_full_name_env:. \\
10152   That~key~will~be~ignored. \\
10153   \c_@@_available_keys_str
10154 }
10155 {
10156   The~available~keys~are~(in~alphabetic~order):~
10157   &~in~blocks,~
10158   ampersand~in~blocks,~
10159   b,~
10160   baseline,~
10161   c,~
10162   cell-space-bottom-limit,~
10163   cell-space-limits,~
10164   cell-space-top-limit,~
10165   code-after,~
10166   code-for-first-col,~

```

```

10167 code-for-first-row,~
10168 code-for-last-col,~
10169 code-for-last-row,~
10170 color-inside,~
10171 columns-type,~
10172 columns-width,~
10173 corners,~
10174 create-extra-nodes,~
10175 create-medium-nodes,~
10176 create-large-nodes,~
10177 extra-left-margin,~
10178 extra-right-margin,~
10179 first-col,~
10180 first-row,~
10181 hlines,~
10182 hvlines,~
10183 hvlines-except-borders,~
10184 l,~
10185 last-col,~
10186 last-row,~
10187 left-margin,~
10188 light-syntax,~
10189 light-syntax-expanded,~
10190 name,~
10191 no-cell-nodes,~
10192 nullify-dots,~
10193 pgf-node-code,~
10194 r,~
10195 renew-dots,~
10196 respect-arraystretch,~
10197 right-margin,~
10198 rounded-corners,~
10199 rules~(with~the~subkeys~'color'~and~'width'),~
10200 small,~
10201 t,~
10202 vlines,~
10203 xdots/color,~
10204 xdots/shorten-start,~
10205 xdots/shorten-end,~
10206 xdots/shorten-and~
10207 xdots/line-style.
10208 }
10209 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10210 {
10211   Unknown~key.\\
10212   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10213   \{NiceTabular\}. \\
10214   That~key~will~be~ignored. \\
10215   \c_@@_available_keys_str
10216 }
10217 {
10218   The~available~keys~are~(in~alphabetic~order):~
10219   &~in~blocks,~
10220   ampersand~in~blocks,~
10221   b,~
10222   baseline,~
10223   c,~
10224   caption,~
10225   cell-space-bottom-limit,~
10226   cell-space-limits,~
10227   cell-space-top-limit,~
10228   code-after,~
10229   code-for-first-col,~

```

```

10230 code-for-first-row,~
10231 code-for-last-col,~
10232 code-for-last-row,~
10233 color-inside,~
10234 columns-width,~
10235 corners,~
10236 custom-line,~
10237 create-extra-nodes,~
10238 create-medium-nodes,~
10239 create-large-nodes,~
10240 extra-left-margin,~
10241 extra-right-margin,~
10242 first-col,~
10243 first-row,~
10244 hlines,~
10245 hvlines,~
10246 hvlines-except-borders,~
10247 label,~
10248 last-col,~
10249 last-row,~
10250 left-margin,~
10251 light-syntax,~
10252 light-syntax-expanded,~
10253 name,~
10254 no-cell-nodes,~
10255 notes~(several~subkeys),~
10256 nullify-dots,~
10257 pgf-node-code,~
10258 renew-dots,~
10259 respect-arraystretch,~
10260 right-margin,~
10261 rounded-corners,~
10262 rules~(with~the~subkeys~'color'~and~'width'),~
10263 short-caption,~
10264 t,~
10265 tabularnote,~
10266 vlines,~
10267 xdots/color,~
10268 xdots/shorten-start,~
10269 xdots/shorten-end,~
10270 xdots/shorten-and~
10271 xdots/line-style.
10272 }
10273 \@@_msg_new:nnn { Duplicate-name }
10274 {
10275 Duplicate-name.\!
10276 The-name-'\

```

```

10293     That~key~will~be~ignored.
10294 }
10295 \@@_msg_new:nn { NiceTabularX~without~X }
10296 {
10297     NiceTabularX~without~X.\\
10298     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10299     However,~you~can~go~on.
10300 }
10301 \@@_msg_new:nn { Preamble~forgotten }
10302 {
10303     Preamble~forgotten.\\
10304     You~have~probably~forgotten~the~preamble~of~your~
10305     \@@_full_name_env:. \\
10306     This~error~is~fatal.
10307 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	We construct the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	91
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
15	After the construction of the array	93
16	We draw the dotted lines	100
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	114
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	142
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	159
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	189
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

31	The keyword <code>\CodeAfter</code>	192
32	The delimiters in the preamble	193
33	The command <code>\SubMatrix</code>	194
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	202
35	The command <code>TikzEveryCell</code>	205
36	The command <code>\ShowCellNames</code>	207
37	We process the options at package loading	209
38	About the package underscore	211
39	Error messages of the package	211