

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

December 16, 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 7.0a of `nicematrix`, at the date of 2024/12/16.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

97 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

We define a command `\iddots` similar to `\ddots` (`'\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.

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

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

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

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

Idem for `\CT@drs@`.

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

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

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

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

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

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`.

```

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

```

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

```

214 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

278 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
293 \dim_new:N \l_@@_col_width_dim
294 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
295 \int_new:N \g_@@_row_total_int
296 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
299 \tl_new:N \l_@@_hpos_cell_tl
300 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
302 \dim_new:N \g_@@_blocks_ht_dim
303 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag.

```
314 \bool_new:N \l_@@_X_bool
315 \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 }`).

```
316 \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.

```
317 \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.

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

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

```
320 \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}`).

```
321 \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`).

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

For `\multicolumn`.

```
323 \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`.

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

```
325 \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 `:`.

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

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

```
328 \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`).

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

```
330 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
331 {
```

```
332   \if_mode_math: \else:
```

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

```
334   \fi:
```

```
335 }
```

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.

```
336 \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”.

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

```
338 \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`).

```
339 \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*.

```

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

342 \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)`.

```

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

```

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

```

349 \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.

```

350 \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.

```

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

```

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

```

355 \bool_new:N \l_@@_in_code_after_bool

```

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

```

356 \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.

```

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

```

359 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```
360 \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.

```
361 \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.

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

```
363 \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.

```
364 \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).

```
365 \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).

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

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

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

```
368 \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).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
384 \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”.

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

```
387 \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!).

```
388 \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}`.

```
389 \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.

```
390 \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`).

```
391 \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`).

```
392 \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.

```
393 \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).

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

```

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

```

The following counters will be used when drawing the rules.

```

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

```

405 \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).

```

406 \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`.

```

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

```

413 \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`.

```

414 \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.

```

415 \dim_new:N \l_@@_offset_dim

```

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

```

416 \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`).

```

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

```

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

```
421 \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).

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

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

```
423 \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).

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

```
425 \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.

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

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

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

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

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

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

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

```
433 \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}`).

```
434 \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.

```
435 \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`).

```
436 \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.

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

```
438 \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.

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

```
440 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

- **Last row**

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

```
441 \int_new:N \l_@@_last_row_int
442 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
444 \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`.

```
445 \int_new:N \l_@@_last_col_int
446 \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.

```
447 \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`).

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

Some utilities

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

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

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

```

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

```

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.

```

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

```

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

```

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

```

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

```

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

```

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.

```

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

```

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.

```
479 \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`.

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

```
484 \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.

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

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

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

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

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

```
493 \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`.

```
494 \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`.

```
495 \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`).

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

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.

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

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`.

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

```

532     }
533   }
534   {
535     \NewDocumentCommand \tabularnote { o m }
536     {
537       \@@_error_or_warning:n { enumitem-not-loaded }
538       \@@_gredirect_none:n { enumitem-not-loaded }
539     }
540   }
541 }

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

```

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

```

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).

```

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

```

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`.

```

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

```

```

576         {
577         \int_if_zero:nTF \l_tmpa_int
578         \c@tabularnote
579         \l_tmpa_int
580         }
581     }
582 }
583 { #1 }
584 }
585 \peek_meaning:Nf \tabularnote
586 {

```

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`.

```

587     \hbox_set:Nn \l_tmpa_box
588     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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`.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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!

```

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

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

```

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

```

```

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

```

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.

```

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

```

7 The options

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

```

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

```

696 \bool_new:N \l_@@_caption_above_bool

```

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

```

697 \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`).

```
698 \dim_new:N \l_@@_cell_space_top_limit_dim
699 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
700 \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.

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

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

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.

```
711 \dim_new:N \l_@@_xdots_radius_dim
712 \hook_gput_code:nnn { begindocument } { . }
713 { \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.

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

```
717 \bool_new:N \l_@@_light_syntax_bool
718 \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).

```
719 \tl_new:N \l_@@_baseline_tl
720 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
721 \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`).

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

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

```
723 \bool_new:N \l_@@_parallelize_diags_bool
724 \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.

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

726 \dim_new:N \l_@@_notes_above_space_dim
727 \hook_gput_code:nnn { begindocument } { . }
728 { \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.).

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

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

```
730 \cs_new_protected:Npn \@@_reset_arraystretch:
731 { \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).

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

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

```
733 \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.

```
734 \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”.

```
735 \bool_new:N \l_@@_medium_nodes_bool
736 \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).

```
737 \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`).

```
738 \dim_new:N \l_@@_left_margin_dim
739 \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`.

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

```
742 \tl_new:N \l_@@_end_of_row_tl
743 \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 “:”.

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

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

```
745 \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 function 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.

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

```
747 \keys_define:nn { nicematrix / xdots }
748 {
749   shorten-start .code:n =
750     \hook_gput_code:nnn { begindocument } { . }
751     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
752   shorten-end .code:n =
753     \hook_gput_code:nnn { begindocument } { . }
754     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
755   shorten-start .value_required:n = true ,
756   shorten-end .value_required:n = true ,
757   shorten .code:n =
758     \hook_gput_code:nnn { begindocument } { . }
759     {
760       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
761       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
762     } ,
763   shorten .value_required:n = true ,
764   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
765   horizontal-labels .default:n = true ,
766   line-style .code:n =
767     {
768       \bool_lazy_or:nnTF
769         { \cs_if_exist_p:N \tikzpicture }
770         { \str_if_eq_p:nn { #1 } { standard } }
771         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
772         { \@@_error:n { bad-option-for~line-style } }
773     } ,
774   line-style .value_required:n = true ,
775   color .tl_set:N = \l_@@_xdots_color_tl ,
776   color .value_required:n = true ,
777   radius .code:n =
778     \hook_gput_code:nnn { begindocument } { . }
779     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
780   radius .value_required:n = true ,
```

```

781   inter .code:n =
782     \hook_gput_code:nnn { begindocument } { . }
783     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
784   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}`.

```

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

```

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

```

```

791 \keys_define:nn { nicematrix / rules }
792 {
793   color .tl_set:N = \l_@@_rules_color_tl ,
794   color .value_required:n = true ,
795   width .dim_set:N = \arrayrulewidth ,
796   width .value_required:n = true ,
797   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
798 }

```

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

```

799 \keys_define:nn { nicematrix / Global }
800 {
801   color-inside .code:n =
802     \@@_warning_gredirect_none:n { key-color-inside } ,
803   colortbl-like .code:n =
804     \@@_warning_gredirect_none:n { key-color-inside } ,
805   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
806   ampersand-in-blocks .default:n = true ,
807   &-in-blocks .meta:n = ampersand-in-blocks ,
808   no-cell-nodes .code:n =
809     \cs_set_protected:Npn \@@_node_for_cell:
810     { \box_use_drop:N \l_@@_cell_box } ,
811   no-cell-nodes .value_forbidden:n = true ,
812   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
813   rounded-corners .default:n = 4 pt ,
814   custom-line .code:n = \@@_custom_line:n { #1 } ,
815   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
816   rules .value_required:n = true ,
817   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
818   standard-cline .default:n = true ,
819   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
820   cell-space-top-limit .value_required:n = true ,
821   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
822   cell-space-bottom-limit .value_required:n = true ,
823   cell-space-limits .meta:n =
824     {
825       cell-space-top-limit = #1 ,
826       cell-space-bottom-limit = #1 ,
827     } ,
828   cell-space-limits .value_required:n = true ,
829   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
830   light-syntax .code:n =

```

```

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

```

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

```

862         { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
863     }
864     { \@@_error:n { One~letter~allowed } }
865 } ,
866 vlines-in-sub-matrix .value_required:n = true ,
867 hvlines .code:n =
868     {
869         \bool_set_true:N \l_@@_hvlines_bool
870         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
871         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
872     } ,
873 hvlines-except-borders .code:n =
874     {
875         \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
876         \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
877         \bool_set_true:N \l_@@_hvlines_bool
878         \bool_set_true:N \l_@@_except_borders_bool
879     } ,
880 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.

```

881     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
882     renew-dots .value_forbidden:n = true ,
883     nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
884     create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
885     create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
886     create-extra-nodes .meta:n =
887         { create-medium-nodes , create-large-nodes } ,
888     left-margin .dim_set:N = \l_@@_left_margin_dim ,

```

```

889 left-margin .default:n = \arraycolsep ,
890 right-margin .dim_set:N = \l_@@_right_margin_dim ,
891 right-margin .default:n = \arraycolsep ,
892 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
893 margin .default:n = \arraycolsep ,
894 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
895 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
896 extra-margin .meta:n =
897   { extra-left-margin = #1 , extra-right-margin = #1 } ,
898 extra-margin .value_required:n = true ,
899 respect-arraystretch .code:n =
900   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
901 respect-arraystretch .value_forbidden:n = true ,
902 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
903 pgf-node-code .value_required:n = true
904 }

```

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

```

905 \keys_define:nn { nicematrix / environments }
906 {
907   corners .clist_set:N = \l_@@_corners_clist ,
908   corners .default:n = { NW , SW , NE , SE } ,
909   code-before .code:n =
910     {
911       \tl_if_empty:nF { #1 }
912       {
913         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
914         \bool_set_true:N \l_@@_code_before_bool
915       }
916     } ,
917   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}`.

```

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

```

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

```

929   \legacy_if:nF { measuring@ }
930   {
931     \str_set:Ne \l_tmpa_str { #1 }
932     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
933     { \@@_error:nn { Duplicate-name } { #1 } }
934     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
935     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
936   } ,
937   name .value_required:n = true ,
938   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
939   code-after .value_required:n = true ,
940 }

```

```

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

```

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

```

986 \keys_define:nn { nicematrix }
987 {
988   NiceMatrixOptions .inherit:n =
989   { nicematrix / Global } ,
990   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
991   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
992   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
993   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
994   SubMatrix / rules .inherit:n = nicematrix / rules ,
995   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
996   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
997   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
998   NiceMatrix .inherit:n =
999   {

```

```

1000     nicematrix / Global ,
1001     nicematrix / environments ,
1002   } ,
1003   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1004   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1005   NiceTabular .inherit:n =
1006     {
1007       nicematrix / Global ,
1008       nicematrix / environments
1009     } ,
1010   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1011   NiceTabular / rules .inherit:n = nicematrix / rules ,
1012   NiceTabular / notes .inherit:n = nicematrix / notes ,
1013   NiceArray .inherit:n =
1014     {
1015       nicematrix / Global ,
1016       nicematrix / environments ,
1017     } ,
1018   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1019   NiceArray / rules .inherit:n = nicematrix / rules ,
1020   pNiceArray .inherit:n =
1021     {
1022       nicematrix / Global ,
1023       nicematrix / environments ,
1024     } ,
1025   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1026   pNiceArray / rules .inherit:n = nicematrix / rules ,
1027 }

```

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

```

1028 \keys_define:nn { nicematrix / NiceMatrixOptions }
1029   {
1030     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1031     delimiters / color .value_required:n = true ,
1032     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1033     delimiters / max-width .default:n = true ,
1034     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1035     delimiters .value_required:n = true ,
1036     width .dim_set:N = \l_@@_width_dim ,
1037     width .value_required:n = true ,
1038     last-col .code:n =
1039       \tl_if_empty:nF { #1 }
1040         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1041       \int_zero:N \l_@@_last_col_int ,
1042     small .bool_set:N = \l_@@_small_bool ,
1043     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.

```

1044     renew-matrix .code:n = \@@_renew_matrix: ,
1045     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}`.

```

1046     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.

```

1047     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)`.

```

1048     \str_if_eq:eeTF { #1 } { auto }
1049     { \@@_error:n { Option~auto~for~columns~width } }
1050     { \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.

```

1051     allow-duplicate-names .code:n =
1052     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1053     allow-duplicate-names .value_forbidden:n = true ,
1054     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1055     notes .value_required:n = true ,
1056     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1057     sub-matrix .value_required:n = true ,
1058     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1059     matrix / columns-type .value_required:n = true ,
1060     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1061     caption-above .default:n = true ,
1062     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1063 }

```

`\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.

```

1064 \NewDocumentCommand \NiceMatrixOptions { m }
1065 { \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.

```

1066 \keys_define:nn { nicematrix / NiceMatrix }
1067 {
1068     last-col .code:n = \tl_if_empty:nTF { #1 }
1069     {
1070         \bool_set_true:N \l_@@_last_col_without_value_bool
1071         \int_set:Nn \l_@@_last_col_int { -1 }
1072     }
1073     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1074     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1075     columns-type .value_required:n = true ,
1076     l .meta:n = { columns-type = l } ,
1077     r .meta:n = { columns-type = r } ,
1078     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1079     delimiters / color .value_required:n = true ,
1080     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1081     delimiters / max-width .default:n = true ,
1082     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1083     delimiters .value_required:n = true ,
1084     small .bool_set:N = \l_@@_small_bool ,
1085     small .value_forbidden:n = true ,
1086     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1087 }

```

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

```

1088 \keys_define:nn { nicematrix / NiceArray }
1089 {

```

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.

```

1090     small .bool_set:N = \l_@@_small_bool ,
1091     small .value_forbidden:n = true ,
1092     last-col .code:n = \tl_if_empty:nF { #1 }
1093         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1094         \int_zero:N \l_@@_last_col_int ,
1095     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1096     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1097     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1098 }
1099 \keys_define:nn { nicematrix / pNiceArray }
1100 {
1101     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1102     last-col .code:n = \tl_if_empty:nF { #1 }
1103         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1104         \int_zero:N \l_@@_last_col_int ,
1105     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1106     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1107     delimiters / color .value_required:n = true ,
1108     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1109     delimiters / max-width .default:n = true ,
1110     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1111     delimiters .value_required:n = true ,
1112     small .bool_set:N = \l_@@_small_bool ,
1113     small .value_forbidden:n = true ,
1114     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1115     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1116     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1117 }

```

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

```

1118 \keys_define:nn { nicematrix / NiceTabular }
1119 {

```

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.

```

1120     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1121         \bool_set_true:N \l_@@_width_used_bool ,
1122     width .value_required:n = true ,
1123     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1124     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1125     tabularnote .value_required:n = true ,
1126     caption .tl_set:N = \l_@@_caption_tl ,
1127     caption .value_required:n = true ,
1128     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1129     short-caption .value_required:n = true ,
1130     label .tl_set:N = \l_@@_label_tl ,
1131     label .value_required:n = true ,
1132     last-col .code:n = \tl_if_empty:nF { #1 }
1133         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1134         \int_zero:N \l_@@_last_col_int ,
1135     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1136     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1137     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1138 }

```

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
```

```

1139 \keys_define:nn { nicematrix / CodeAfter }
1140 {
1141   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1142   delimiters / color .value_required:n = true ,
1143   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1144   rules .value_required:n = true ,
1145   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1146   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1147   sub-matrix .value_required:n = true ,
1148   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1149 }

```

8 Important code used by {NiceArrayWithDelims}

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

```

1150 \cs_new_protected:Npn \@@_cell_begin:
1151 {

```

`\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.

```
1152   \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).

```
1153   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1154   \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.

```

1155   \int_compare:nNnT \c@jCol = \c_one_int
1156     { \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:`.

```
1157   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1158   \@@_tuning_not_tabular_begin:
1159   \@@_tuning_first_row:
1160   \@@_tuning_last_row:
1161   \g_@@_row_style_tl
1162 }

```

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.

```

1163 \cs_new_protected:Npn \@@_tuning_first_row:
1164 {
1165     \if_int_compare:w \c@iRow = \c_zero_int
1166         \if_int_compare:w \c@jCol > \c_zero_int
1167             \l_@@_code_for_first_row_tl
1168             \xglobal \colorlet { nicematrix-first-row } { . }
1169         \fi:
1170     \fi:
1171 }

```

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.

```

1172 \cs_new_protected:Npn \@@_tuning_last_row:
1173 {
1174     \if_int_compare:w \c@iRow = \l_@@_last_row_int
1175         \l_@@_code_for_last_row_tl
1176         \xglobal \colorlet { nicematrix-last-row } { . }
1177     \fi:
1178 }

```

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

```

1179 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1180 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1181 {
1182     \m@th % added 2024/11/21
1183     \c_math_toggle_token

```

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

```

1184     \@@_tuning_key_small:
1185 }
1186 \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.

```

1187 \cs_new_protected:Npn \@@_begin_of_row:
1188 {
1189     \int_gincr:N \c@iRow
1190     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1191     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1192     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1193     \pgfpicture
1194     \pgfrememberpicturepositiononpagetrue
1195     \pgfcoordinate
1196     { \@@_env: - row - \int_use:N \c@iRow - base }

```

```

1197     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1198 \str_if_empty:NF \l_@@_name_str
1199 {
1200     \pgfnodealias
1201     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1202     { \@@_env: - row - \int_use:N \c@iRow - base }
1203 }
1204 \endpgfpicture
1205 }

```

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.

```

1206 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1207 {
1208     \int_if_zero:nTF \c@iRow
1209     {
1210         \dim_compare:nNnT { \box_dp:N \l_@@_cell_box } > \g_@@_dp_row_zero_dim
1211         { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1212         \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_zero_dim
1213         { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1214     }
1215     {
1216         \int_compare:nNnT \c@iRow = \c_one_int
1217         {
1218             \dim_compare:nNnT { \box_ht:N \l_@@_cell_box } > \g_@@_ht_row_one_dim
1219             { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1220         }
1221     }
1222 }
1223 \cs_new_protected:Npn \@@_rotate_cell_box:
1224 {
1225     \box_rotate:Nn \l_@@_cell_box { 90 }
1226     \bool_if:NTF \g_@@_rotate_c_bool
1227     {
1228         \hbox_set:Nn \l_@@_cell_box
1229         {
1230             \m@th % add 2024/11/21
1231             \c_math_toggle_token
1232             \vcenter { \box_use:N \l_@@_cell_box }
1233             \c_math_toggle_token
1234         }
1235     }
1236     {
1237         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1238         {
1239             \vbox_set_top:Nn \l_@@_cell_box
1240             {
1241                 \vbox_to_zero:n { }
1242                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1243                 \box_use:N \l_@@_cell_box
1244             }
1245         }
1246     }
1247     \bool_gset_false:N \g_@@_rotate_bool
1248     \bool_gset_false:N \g_@@_rotate_c_bool
1249 }
1250 \cs_new_protected:Npn \@@_adjust_size_box:
1251 {

```

```

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

```

The following command is nullified in the tabulars.

```

1273 \@@_tuning_not_tabular_end:
1274 \hbox_set_end:
1275 \@@_cell_end_i:
1276 }
1277 \cs_new_protected:Npn \@@_cell_end_i:
1278 {

```

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.

```

1279 \g_@@_cell_after_hook_tl
1280 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1281 \@@_adjust_size_box:
1282 \box_set_ht:Nn \l_@@_cell_box
1283 { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1284 \box_set_dp:Nn \l_@@_cell_box
1285 { \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”).

```

1286 \@@_update_max_cell_width:

```

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

```

1287 \@@_update_for_first_and_last_row:

```

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

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

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

- for the columns of type p, m, b, V (of varwidth) or X, we test whether the cell is syntactically empty with `\@@_test_if_empty:` and `\@@_test_if_empty_for_S:`
- if the width of the box `\l_@@_cell_box` (created with the content of the cell) is equal to zero, we consider the cell as empty (however, this is not perfect since the user may have used a `\rlap`, `\llap`, `\clap` or a `\mathclap` of `mathtools`).

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

```

1288 \bool_if:NTF \g_@@_empty_cell_bool
1289 { \box_use_drop:N \l_@@_cell_box }
1290 {
1291   \bool_if:NTF \g_@@_not_empty_cell_bool
1292   \@@_node_for_cell:
1293   {
1294     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1295     \@@_node_for_cell:
1296     { \box_use_drop:N \l_@@_cell_box }
1297   }
1298 }
1299 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1300 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1301 \bool_gset_false:N \g_@@_empty_cell_bool
1302 \bool_gset_false:N \g_@@_not_empty_cell_bool
1303 }

```

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

```

1304 \cs_new_protected:Npn \@@_update_max_cell_width:
1305 {
1306   \dim_gset:Nn \g_@@_max_cell_width_dim
1307   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1308 }

```

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`).

```

1309 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1310 {
1311   \@@_math_toggle:
1312   \hbox_set_end:
1313   \bool_if:NF \g_@@_rotate_bool
1314   {
1315     \hbox_set:Nn \l_@@_cell_box
1316     {
1317       \makebox [ \l_@@_col_width_dim ] [ s ]
1318       { \hbox_unpack_drop:N \l_@@_cell_box }
1319     }
1320   }
1321   \@@_cell_end_i:
1322 }

```

```

1323 \pgfset
1324 {
1325   nicematrix / cell-node /.style =
1326   {
1327     inner-sep = \c_zero_dim ,
1328     minimum-width = \c_zero_dim
1329   }
1330 }

```

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

```

1331 \cs_new_protected:Npn \@@_node_for_cell:
1332 {

```

```

1333 \pgfpicture
1334 \pgfsetbaseline \c_zero_dim
1335 \pgfrememberpicturepositiononpagetrue
1336 \pgfset { nicematrix / cell-node }
1337 \pgfnode
1338   { rectangle }
1339   { base }
1340   {

```

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).

```

1341     \set@color
1342     \box_use_drop:N \l_@@_cell_box
1343   }
1344   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1345   { \l_@@_pgf_node_code_tl }
1346 \str_if_empty:NF \l_@@_name_str
1347   {
1348     \pgfnodealias
1349     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1350     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1351   }
1352 \endpgfpicture
1353 }

```

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.

```

1354 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1355 {
1356   \cs_new_protected:Npn \@@_patch_node_for_cell:
1357   {
1358     \hbox_set:Nn \l_@@_cell_box
1359     {
1360       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1361       \hbox_overlap_left:n
1362       {
1363         \pgfsys@markposition
1364         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

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

```

1365         #1
1366       }
1367     \box_use:N \l_@@_cell_box
1368     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1369     \hbox_overlap_left:n
1370     {
1371       \pgfsys@markposition
1372       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1373     }
1374   }
1375 }
1376 }
1377 }

```

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

```

1378 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1379 {
1380   \@@_patch_node_for_cell:n
1381   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1382 }
1383 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

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

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

the content of `\g_@@_Cdots_lines_tl` will be:

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

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

```
1384 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1385 {
1386   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1387   { \g_@@_#2 _ lines _ tl }
1388   {
1389     \use:c { @@ _ draw _ #2 : nnn }
1390     { \int_use:N \c@iRow }
1391     { \int_use:N \c@jCol }
1392     { \exp_not:n { #3 } }
1393   }
1394 }

1395 \cs_generate_variant:Nn \@@_array:n { o }
1396 \cs_new_protected:Npn \@@_array:n
1397 {
1398   % \begin{macrocode}
1399   \dim_set:Nn \col@sep
1400   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1401   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1402   { \cs_set_nopar:Npn \@halignto { } }
1403   { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1404   \@tabarray
\l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose
the \array (of array) with the option t and the right translation will be done further. Re-
mark that \str_if_eq:eeTF is fully expandable and we need something fully expandable here.
\str_if_eq:ee(TF) is faster than \str_if_eq:nn(TF).
1405   [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1406 }
```

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`.

```
1407 \bool_if:nTF
1408 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1409 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1410 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```

1411 \cs_new_protected:Npn \@@_create_row_node:
1412 {
1413   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1414     {
1415       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1416       \@@_create_row_node_i:
1417     }
1418 }
1419 \cs_new_protected:Npn \@@_create_row_node_i:
1420 {

```

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

```

1421   \hbox
1422   {
1423     \bool_if:NT \l_@@_code_before_bool
1424     {
1425       \vtop
1426       {
1427         \skip_vertical:N 0.5\arrayrulewidth
1428         \pgfsys@markposition
1429         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1430         \skip_vertical:N -0.5\arrayrulewidth
1431       }
1432     }
1433     \pgfpicture
1434     \pgfrememberpicturepositiononpagetrue
1435     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1436     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1437     \str_if_empty:NF \l_@@_name_str
1438     {
1439       \pgfnodealias
1440       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1441       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1442     }
1443     \endpgfpicture
1444   }
1445 }

```

```

1446 \cs_new_protected:Npn \@@_in_everycr:
1447 {
1448   \bool_if:NT \c_@@_recent_array_bool
1449   {
1450     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1451     \tbl_update_cell_data_for_next_row:
1452   }
1453   \int_gzero:N \c@jCol
1454   \bool_gset_false:N \g_@@_after_col_zero_bool
1455   \bool_if:NF \g_@@_row_of_col_done_bool
1456   {
1457     \@@_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).

```

1458     \clist_if_empty:NF \l_@@_hlines_clist
1459     {
1460       \str_if_eq:eeF \l_@@_hlines_clist { all }
1461       {
1462         \clist_if_in:NeT
1463         \l_@@_hlines_clist
1464         { \int_eval:n { \c@iRow + 1 } }
1465       }
1466     }

```

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.

```

1467         \int_compare:nNnT \c@iRow > { -1 }
1468         {
1469             \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1470             { \hrule height \arrayrulewidth width \c_zero_dim }
1471         }
1472     }
1473 }
1474 }
1475 }

```

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

```

1476 \cs_set_protected:Npn \@@_renew_dots:
1477 {
1478     \cs_set_eq:NN \ldots \@@_Ldots
1479     \cs_set_eq:NN \cdots \@@_Cdots
1480     \cs_set_eq:NN \vdots \@@_Vdots
1481     \cs_set_eq:NN \ddots \@@_Ddots
1482     \cs_set_eq:NN \iddots \@@_Iddots
1483     \cs_set_eq:NN \dots \@@_Ldots
1484     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1485 }

```

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

```

1486 \hook_gput_code:nnn { begindocument } { . }
1487 {
1488     \IfPackageLoadedTF { colortbl }
1489     {
1490         \cs_set_protected:Npn \@@_everycr:
1491         { \CT@everycr { \noalign { \@@_in_everycr: } } }
1492     }
1493     {
1494         \cs_new_protected:Npn \@@_everycr:
1495         { \everycr { \noalign { \@@_in_everycr: } } }
1496     }
1497 }

```

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.

```

1498 \hook_gput_code:nnn { begindocument } { . }
1499 {
1500     \IfPackageLoadedTF { booktabs }
1501     {
1502         \cs_new_protected:Npn \@@_patch_booktabs:
1503         { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1504     }
1505     { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1506 }

```

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`

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}`.

(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`.

```

1507 \cs_new_protected:Npn \@@_some_initialization:
1508   {
1509     \@@_everycr:
1510     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1511     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1512     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1513     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1514     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1515     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1516   }

```

```

1517 \cs_new_protected:Npn \@@_pre_array_ii:
1518   {

```

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

```

1519     \int_gzero:N \g_@@_total_X_weight_int
1520     \@@_expand_clist:N \l_@@_hlines_clist
1521     \@@_expand_clist:N \l_@@_vlines_clist
1522     \@@_patch_booktabs:
1523     \box_clear_new:N \l_@@_cell_box
1524     \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}`).

```

1525     \bool_if:NT \l_@@_small_bool
1526     {
1527         \cs_set_nopar:Npn \arraystretch { 0.47 }
1528         \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1529         \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1530     }

```

```

1531     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1532     {
1533         \tl_put_right:Nn \@@_begin_of_row:
1534         {
1535             \pgfsys@markposition
1536             { \@@_env: - row - \int_use:N \c@iRow - base }
1537         }
1538     }

```

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

```

1539     \bool_if:nTF
1540     { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1541     {
1542         \cs_set_nopar:Npn \ar@ialign
1543         {
1544             \bool_if:NT \c_@@_testphase_table_bool
1545             \tbl_init_cell_data_for_table:
1546             \@@_some_initialization:
1547             \dim_zero:N \tabskip

```

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

```

1548         \cs_set_eq:NN \ar@ialign \@_old_ar@ialign:
1549         \halign
1550     }
1551 }

```

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

```

1552 {
1553     \cs_set_nopar:Npn \ialign
1554     {
1555         \@_some_initialization:
1556         \dim_zero:N \tabskip
1557         \cs_set_eq:NN \ialign \@_old_ialign:
1558         \halign
1559     }
1560 }

```

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

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

```

1561     \bool_if:NT \c_@@_revtex_bool
1562     {
1563         \IfPackageLoadedT { colortbl }
1564         { \cs_set_protected:Npn \CT@setup { } }
1565     }

```

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

```

1566     \cs_set_eq:NN \@_old_ldots \ldots
1567     \cs_set_eq:NN \@_old_cdots \cdots
1568     \cs_set_eq:NN \@_old_vdots \vdots
1569     \cs_set_eq:NN \@_old_ddots \ddots
1570     \cs_set_eq:NN \@_old_iddots \iddots
1571     \bool_if:NTF \l_@@_standard_cline_bool
1572     { \cs_set_eq:NN \cline \@_standard_cline }
1573     { \cs_set_eq:NN \cline \@_cline }
1574     \cs_set_eq:NN \Ldots \@_Ldots
1575     \cs_set_eq:NN \Cdots \@_Cdots
1576     \cs_set_eq:NN \Vdots \@_Vdots
1577     \cs_set_eq:NN \Ddots \@_Ddots
1578     \cs_set_eq:NN \Iddots \@_Iddots
1579     \cs_set_eq:NN \Hline \@_Hline:
1580     \cs_set_eq:NN \Hspace \@_Hspace:
1581     \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1582     \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1583     \cs_set_eq:NN \Block \@_Block:
1584     \cs_set_eq:NN \rotate \@_rotate:
1585     \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1586     \cs_set_eq:NN \dotfill \@_dotfill:
1587     \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1588     \cs_set_eq:NN \diagbox \@_diagbox:nn
1589     \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1590     \cs_set_eq:NN \RowStyle \@_RowStyle:n
1591     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1592     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1593     \cs_set_eq:NN \cellcolor \@_cellcolor_tabular
1594     \cs_set_eq:NN \rowcolor \@_rowcolor_tabular

```

```

1595 \cs_set_eq:NN \rowcolors \@_rowcolors_tabular
1596 \cs_set_eq:NN \rowlistcolors \@_rowlistcolors_tabular
1597 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1598   { \cs_set_eq:NN \@_tuning_first_row: \prg_do_nothing: }
1599 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1600   { \cs_set_eq:NN \@_tuning_last_row: \prg_do_nothing: }
1601 \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:`.

```

1602 \cs_set_eq:NN \multicolumn \@_multicolumn:nnn
1603 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1604   { \cs_set_eq:NN \multicolumn \@_old_multicolumn }
1605 \@_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!).

```

1606 \tl_if_exist:NT \l_@@_note_in_caption_tl
1607   {
1608     \tl_if_empty:NF \l_@@_note_in_caption_tl
1609     {
1610       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1611       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1612     }
1613   }

```

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).

```

1614 \seq_gclear:N \g_@@_multicolumn_cells_seq
1615 \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).

```

1616 \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`).

```

1617 \int_gzero_new:N \g_@@_row_total_int

```

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

```

1618 \int_gzero_new:N \g_@@_col_total_int
1619 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1620 \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.

```

1621 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1622 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1623 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1624 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1625 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1626 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

```

```

1627 \tl_gclear:N \g_nicematrix_code_before_tl
1628 \tl_gclear:N \g_@@_pre_code_before_tl
1629 }

```

This is the end of \@@_pre_array_ii:.

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

```

1630 \cs_new_protected:Npn \@@_pre_array:
1631 {
1632 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1633 \int_gzero_new:N \c@iRow
1634 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1635 \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).

```

1636 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1637 {
1638 \bool_set_true:N \l_@@_last_row_without_value_bool
1639 \bool_if:NT \g_@@_aux_found_bool
1640 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1641 }
1642 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1643 {
1644 \bool_if:NT \g_@@_aux_found_bool
1645 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1646 }

```

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

```

1647 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1648 {
1649 \tl_put_right:Nn \@@_update_for_first_and_last_row:
1650 {
1651 \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1652 { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1653 \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1654 { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1655 }
1656 }

```

```

1657 \seq_gclear:N \g_@@_cols_vlism_seq
1658 \seq_gclear:N \g_@@_submatrix_seq

```

Now the \CodeBefore.

```

1659 \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.

```

1660 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1661 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1662 \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.

```
1663 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1664 \@@_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.

```
1665 \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]`).

```
1666 \dim_zero_new:N \l_@@_left_delim_dim
1667 \dim_zero_new:N \l_@@_right_delim_dim
1668 \bool_if:NTF \g_@@_delims_bool
1669 {
```

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

```
1670 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1671 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1672 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1673 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1674 }
1675 {
1676 \dim_gset:Nn \l_@@_left_delim_dim
1677 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1678 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1679 }
```

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).

```
1680 \hbox_set:Nw \l_@@_the_array_box
1681 \skip_horizontal:N \l_@@_left_margin_dim
1682 \skip_horizontal:N \l_@@_extra_left_margin_dim
1683 \bool_if:NT \c_@@_recent_array_bool
1684 { \UseTaggingSocket { tbl / hmode / begin } }
```

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

```
1685 \m@th
1686 \c_math_toggle_token
1687 \bool_if:NTF \l_@@_light_syntax_bool
1688 { \use:c { @@-light-syntax } }
1689 { \use:c { @@-normal-syntax } }
1690 }
```

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

```
1691 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1692 {
1693 \tl_set:Nn \l_tmpa_tl { #1 }
1694 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1695 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1696 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1697 \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...

```
1698   \@@_pre_array:
1699   }
```

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).

```
1700 \cs_new_protected:Npn \@@_pre_code_before:
1701   {
```

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).

```
1702   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1703   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1704   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1705   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

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

```
1706   \pgfsys@markposition { \@@_env: - position }
1707   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1708   \pgfpicture
1709   \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1710   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1711   {
1712     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1713     \pgfcoordinate { \@@_env: - row - ##1 }
1714     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1715   }
```

Now, the recreation of the `col` nodes.

```
1716   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1717   {
1718     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1719     \pgfcoordinate { \@@_env: - col - ##1 }
1720     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1721   }
```

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

```
1722   \@@_create_diag_nodes:
```

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

```
1723   \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1724   \endpgfpicture
```

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

```
1725   \@@_create_blocks_nodes:
```

```

1726 \IfPackageLoadedT { tikz }
1727 {
1728   \tikzset
1729   {
1730     every-picture / .style =
1731     { overlay , name-prefix = \@@_env: - }
1732   }
1733 }
1734 \cs_set_eq:NN \cellcolor \@@_cellcolor
1735 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1736 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1737 \cs_set_eq:NN \rowcolor \@@_rowcolor
1738 \cs_set_eq:NN \rowcolors \@@_rowcolors
1739 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1740 \cs_set_eq:NN \arraycolor \@@_arraycolor
1741 \cs_set_eq:NN \columncolor \@@_columncolor
1742 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1743 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1744 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1745 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1746 }

```

```

1747 \cs_new_protected:Npn \@@_exec_code_before:
1748 {

```

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...

```

1749 \clist_map_inline:Nn \l_@@_corners_cells_clist
1750 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1751 \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`.

```

1752 \@@_add_to_colors_seq:nn { { nocolor } } { }
1753 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1754 \group_begin:

```

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

```

1755 \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`).

```

1756 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1757 { \@@_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`.

```

1758 \exp_last_unbraced:No \@@_CodeBefore_keys:
1759 \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.

```

1760 \@@_actually_color:
1761 \l_@@_code_before_tl
1762 \q_stop

```

```

1763 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1764 \group_end:
1765 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1766 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1767 }

```

```

1768 \keys_define:nn { nicematrix / CodeBefore }
1769 {
1770   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1771   create-cell-nodes .default:n = true ,
1772   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1773   sub-matrix .value_required:n = true ,
1774   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1775   delimiters / color .value_required:n = true ,
1776   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1777 }
1778 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1779 {
1780   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1781   \@@_CodeBefore:w
1782 }

```

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.

```

1783 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1784 {
1785   \bool_if:NT \g_@@_aux_found_bool
1786   {
1787     \@@_pre_code_before:
1788     #1
1789   }
1790 }

```

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.

```

1791 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1792 {
1793   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1794   {
1795     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1796     \pgfcoordinate { \@@_env: - row - ##1 - base }
1797     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1798     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1799     {
1800       \cs_if_exist:cT
1801       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1802       {
1803         \pgfsys@getposition
1804         { \@@_env: - ##1 - #####1 - NW }
1805         \@@_node_position:
1806         \pgfsys@getposition
1807         { \@@_env: - ##1 - #####1 - SE }
1808         \@@_node_position_i:
1809         \@@_pgf_rect_node:nnn
1810         { \@@_env: - ##1 - #####1 }
1811         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1812         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1813       }
1814     }

```

```

1815     }
1816 \int_step_inline:nn \c@iRow
1817 {
1818     \pgfnodealias
1819     { \@@_env: - ##1 - last }
1820     { \@@_env: - ##1 - \int_use:N \c@jCol }
1821 }
1822 \int_step_inline:nn \c@jCol
1823 {
1824     \pgfnodealias
1825     { \@@_env: - last - ##1 }
1826     { \@@_env: - \int_use:N \c@iRow - ##1 }
1827 }
1828 \@@_create_extra_nodes:
1829 }

```

```

1830 \cs_new_protected:Npn \@@_create_blocks_nodes:
1831 {
1832     \pgfpicture
1833     \pgf@relevantforpicturesizefalse
1834     \pgfrememberpicturepositiononpagetrue
1835     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1836     { \@@_create_one_block_node:nnnnn ##1 }
1837     \endpgfpicture
1838 }

```

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

```

1839 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1840 {
1841     \tl_if_empty:nF { #5 }
1842     {
1843         \@@_qpoint:n { col - #2 }
1844         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1845         \@@_qpoint:n { #1 }
1846         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1847         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1848         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1849         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1850         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1851         \@@_pgf_rect_node:nnnnn
1852         { \@@_env: - #5 }
1853         { \dim_use:N \l_tmpa_dim }
1854         { \dim_use:N \l_tmpb_dim }
1855         { \dim_use:N \l_@@_tmpc_dim }
1856         { \dim_use:N \l_@@_tmpd_dim }
1857     }
1858 }

```

```

1859 \cs_new_protected:Npn \@@_patch_for_revtext:
1860 {
1861     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1862     \cs_set_eq:NN \@array \@array@array
1863     \cs_set_eq:NN \@tabular \@tabular@array
1864     \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1865     \cs_set_eq:NN \array \array@array
1866     \cs_set_eq:NN \endarray \endarray@array
1867     \cs_set:Npn \endtabular { \endarray $\egroup} % $

```

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

```

1868 \cs_set_eq:NN \@mkpream \@mkpream@array
1869 \cs_set_eq:NN \@classx \@classx@array
1870 \cs_set_eq:NN \insert@column \insert@column@array
1871 \cs_set_eq:NN \@arraycr \@arraycr@array
1872 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1873 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1874 }

```

10 The environment `{NiceArrayWithDelims}`

```

1875 \NewDocumentEnvironment { NiceArrayWithDelims }
1876 { m m 0 { } m ! 0 { } t \CodeBefore }
1877 {
1878 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1879 \@@_provide_pgfsyspdfmark:
1880 \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

1881 \bgroup

1882 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1883 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1884 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1885 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1886 \int_gzero:N \g_@@_block_box_int
1887 \dim_zero:N \g_@@_width_last_col_dim
1888 \dim_zero:N \g_@@_width_first_col_dim
1889 \bool_gset_false:N \g_@@_row_of_col_done_bool
1890 \str_if_empty:NT \g_@@_name_env_str
1891 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1892 \bool_if:NTF \l_@@_tabular_bool
1893 \mode_leave_vertical:
1894 \@@_test_if_math_mode:
1895 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet-in-env } }
1896 \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.

```

1897 \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.

```

1898 \cs_if_exist:NT \tikz@library@external@loaded
1899 {
1900 \tikzexternaldisable
1901 \cs_if_exist:NT \ifstandalone
1902 { \tikzset { external / optimize = false } }
1903 }

```

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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1904 \int_gincr:N \g_@@_env_int
1905 \bool_if:NF \l_@@_block_auto_columns_width_bool
1906 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1907 \seq_gclear:N \g_@@_blocks_seq
1908 \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`.

```

1909 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1910 \seq_gclear:N \g_@@_pos_of_xdots_seq
1911 \tl_gclear_new:N \g_@@_code_before_tl
1912 \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.

```

1913 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1914 {
1915   \bool_gset_true:N \g_@@_aux_found_bool
1916   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1917 }
1918 { \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.

```

1919 \tl_gclear:N \g_@@_aux_tl
1920 \tl_if_empty:NF \g_@@_code_before_tl
1921 {
1922   \bool_set_true:N \l_@@_code_before_bool
1923   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1924 }
1925 \tl_if_empty:NF \g_@@_pre_code_before_tl
1926 { \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`.

```

1927 \bool_if:NTF \g_@@_delims_bool
1928 { \keys_set:nn { nicematrix / pNiceArray } }
1929 { \keys_set:nn { nicematrix / NiceArray } }
1930 { #3 , #5 }

```

```

1931 \@@_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:.`

```

1932 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1933 }

```

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

```

1934 {
1935   \bool_if:NTF \l_@@_light_syntax_bool
1936   { \use:c { end @@-light-syntax } }
1937   { \use:c { end @@-normal-syntax } }
1938   \c_math_toggle_token
1939   \skip_horizontal:N \l_@@_right_margin_dim
1940   \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1941
1942 % awful workaround
1943 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1944 {
1945     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1946     {
1947         \skip_horizontal:N - \l_@@_columns_width_dim
1948         \bool_if:NTF \l_@@_tabular_bool
1949             { \skip_horizontal:n { - 2 \tabcolsep } }
1950             { \skip_horizontal:n { - 2 \arraycolsep } }
1951     }
1952 }
1953 \hbox_set_end:
1954 \bool_if:NT \c_@@_recent_array_bool
1955     { \UseTaggingSocket { tbl / hmode / end } }

```

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

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

```

1956 \bool_if:NT \l_@@_width_used_bool
1957 {
1958     \int_if_zero:nT \g_@@_total_X_weight_int
1959     { \@@_error_or_warning:n { width-without-X~columns } }
1960 }

```

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 .

```

1961 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1962     { \@@_compute_width_X: }

```

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

```

1963 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1964 {
1965     \bool_if:NF \l_@@_last_row_without_value_bool
1966     {
1967         \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1968         {
1969             \@@_error:n { Wrong~last-row }
1970             \int_gset_eq:NN \l_@@_last_row_int \c_iRow
1971         }
1972     }
1973 }

```

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

```

1974 \int_gset_eq:NN \c_jCol \g_@@_col_total_int
1975 \bool_if:NTF \g_@@_last_col_found_bool
1976     { \int_gdecr:N \c_jCol }
1977 {
1978     \int_compare:nNnT \l_@@_last_col_int > { -1 }
1979     { \@@_error:n { last~col~not~used } }
1980 }

```

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

```

1981 \int_gset_eq:NN \g_@@_row_total_int \c_iRow
1982 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c_iRow }

```

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

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).

```
1983 \int_if_zero:nTF \l_@@_first_col_int
1984 { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
1985 \bool_if:nTF { ! \g_@@_delims_bool }
1986 {
1987   \str_if_eq:eeTF \l_@@_baseline_tl { c }
1988   \@@_use_arraybox_with_notes_c:
1989   {
1990     \str_if_eq:eeTF \l_@@_baseline_tl { b }
1991     \@@_use_arraybox_with_notes_b:
1992     \@@_use_arraybox_with_notes:
1993   }
1994 }
```

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).

```
1995 {
1996   \int_if_zero:nTF \l_@@_first_row_int
1997   {
1998     \dim_set:eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1999     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2000   }
2001   { \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

```
2002 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2003 {
2004   \dim_set:eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2005   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2006 }
2007 { \dim_zero:N \l_tmpb_dim }
2008 \hbox_set:Nn \l_tmpa_box
2009 {
2010   \m@th % added 2024/11/21
2011   \c_math_toggle_token
2012   \@@_color:o \l_@@_delimiters_color_tl
2013   \exp_after:wN \left \g_@@_left_delim_tl
2014   \vcenter
2015   {
```

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.

```
2016   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2017   \hbox
2018   {
2019     \bool_if:NTF \l_@@_tabular_bool
2020     { \skip_horizontal:N -\tabcolsep }
2021     { \skip_horizontal:N -\arraycolsep }
2022     \@@_use_arraybox_with_notes_c:
2023     \bool_if:NTF \l_@@_tabular_bool
2024     { \skip_horizontal:N -\tabcolsep }
2025     { \skip_horizontal:N -\arraycolsep }
2026   }
```

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

```
2027   \skip_vertical:N -\l_tmpb_dim
2028   \skip_vertical:N \arrayrulewidth
```

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).

```

2029     }
2030     \exp_after:wN \right \g_@@_right_delim_tl
2031     \c_math_toggle_token
2032   }

```

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.

```

2033     \bool_if:NTF \l_@@_delimiters_max_width_bool
2034     {
2035       \@@_put_box_in_flow_bis:nn
2036       \g_@@_left_delim_tl
2037       \g_@@_right_delim_tl
2038     }
2039     \@@_put_box_in_flow:
2040   }

```

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).

```

2041     \bool_if:NT \g_@@_last_col_found_bool
2042     { \skip_horizontal:N \g_@@_width_last_col_dim }
2043     \bool_if:NT \l_@@_preamble_bool
2044     {
2045       \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2046       { \@@_warning_gredirect_none:n { columns-not-used } }
2047     }
2048     \@@_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.

```

2049     \egroup

```

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

```

2050     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2051     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2052     \iow_now:Ne \@mainaux
2053     {
2054       \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2055       { \exp_not:o \g_@@_aux_tl }
2056     }
2057     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2058     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2059   }

```

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

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

```

2060 \cs_new_protected:Npn \@@_compute_width_X:
2061 {
2062   \tl_gput_right:Ne \g_@@_aux_tl
2063   {
2064     \bool_set_true:N \l_@@_X_columns_aux_bool
2065     \dim_set:Nn \l_@@_X_columns_dim
2066     {
2067       \dim_compare:nNnTF
2068       {
2069         \dim_abs:n
2070         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2071       }
2072       <

```

```

2073     { 0.001 pt }
2074     { \dim_use:N \l_@@_X_columns_dim }
2075     {
2076       \dim_eval:n
2077       {
2078         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2079         / \int_use:N \g_@@_total_X_weight_int
2080         + \l_@@_X_columns_dim
2081       }
2082     }
2083   }
2084 }
2085 }

```

11 Construction of the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble which will be given to `{array}` (of the package `array`).

The preamble given by the final user is stored in `\g_@@_user_preamble_tl`. The modified version will be stored in `\g_@@_array_preamble_tl` also.

```

2086 \cs_new_protected:Npn \@@_transform_preamble:
2087 {
2088   \@@_transform_preamble_i:
2089   \@@_transform_preamble_ii:
2090 }
2091 \cs_new_protected:Npn \@@_transform_preamble_i:
2092 {
2093   \int_gzero:N \c@jCol

```

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

```

2094   \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.

```

2095   \bool_gset_false:N \g_tmpb_bool

```

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

```

2096   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2097   \int_zero:N \l_tmpa_int
2098   \tl_gclear:N \g_@@_array_preamble_tl
2099   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2100   {
2101     \tl_gset:Nn \g_@@_array_preamble_tl
2102     { ! { \skip_horizontal:N \arrayrulewidth } }
2103   }
2104   {
2105     \clist_if_in:NnT \l_@@_vlines_clist 1
2106     {
2107       \tl_gset:Nn \g_@@_array_preamble_tl
2108       { ! { \skip_horizontal:N \arrayrulewidth } }
2109     }
2110   }

```

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

```

2111   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2112   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2113 \@@_replace_columncolor:
2114 }

```

```

2115 \hook_gput_code:nnn { begindocument } { . }
2116 {
2117   \IfPackageLoadedTF { colortbl }
2118   {

```

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).

```

2119     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2120     \cs_new_protected:Npn \@@_replace_columncolor:
2121     {
2122       \regex_replace_all:NnN
2123       \c_@@_columncolor_regex
2124       { \c { @@_columncolor_preamble } }
2125       \g_@@_array_preamble_tl
2126     }
2127   }
2128   {
2129     \cs_new_protected:Npn \@@_replace_columncolor:
2130     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2131   }
2132 }

```

```

2133 \cs_new_protected:Npn \@@_transform_preamble_ii:
2134 {

```

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

```

2135   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2136   {
2137     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2138     { \bool_gset_true:N \g_@@_delims_bool }
2139   }
2140   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2141   \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).

```

2142   \int_if_zero:nTF \l_@@_first_col_int
2143   { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2144   {
2145     \bool_if:NF \g_@@_delims_bool
2146     {
2147       \bool_if:NF \l_@@_tabular_bool
2148       {
2149         \clist_if_empty:NT \l_@@_vlines_clist
2150         {
2151           \bool_if:NF \l_@@_exterior_arraycolsep_bool
2152           { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2153         }
2154       }
2155     }
2156   }
2157   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2158   { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_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_right:Nn \g_@@_array_preamble_tl { @ { } } }
2168     }
2169     }
2170     }
2171     }

```

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`).

```

2172 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2173 {

```

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

```

2174 \bool_if:NF \c_@@_testphase_table_bool
2175 {
2176 \tl_gput_right:Nn \g_@@_array_preamble_tl
2177 { > { \@@_error_too_much_cols: } l }
2178 }
2179 }
2180 }

```

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.

```

2181 \cs_new_protected:Npn \@@_rec_preamble:n #1
2182 {

```

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

```

2183 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2184 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2185 {

```

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

```

2186 \cs_if_exist:cTF { NC @ find @ #1 }
2187 {
2188 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2189 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2190 }
2191 {
2192 \str_if_eq:nnTF { #1 } { S }
2193 { \@@_fatal:n { unknown~column~type~S } }
2194 { \@@_fatal:nn { unknown~column~type } { #1 } }
2195 }
2196 }
2197 }

```

For c, l and r

```

2198 \cs_new_protected:Npn \@@_c #1
2199 {
2200 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl

```

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

```

2201 \tl_gclear:N \g_@@_pre_cell_tl
2202 \tl_gput_right:Nn \g_@@_array_preamble_tl
2203 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2204 \int_gincr:N \c@jCol
2205 \@@_rec_preamble_after_col:n
2206 }

2207 \cs_new_protected:Npn \@@_l #1
2208 {
2209 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2210 \tl_gclear:N \g_@@_pre_cell_tl
2211 \tl_gput_right:Nn \g_@@_array_preamble_tl
2212 {
2213 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2214 l
2215 < \@@_cell_end:
2216 }
2217 \int_gincr:N \c@jCol
2218 \@@_rec_preamble_after_col:n
2219 }

2220 \cs_new_protected:Npn \@@_r #1
2221 {
2222 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2223 \tl_gclear:N \g_@@_pre_cell_tl
2224 \tl_gput_right:Nn \g_@@_array_preamble_tl
2225 {
2226 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2227 r
2228 < \@@_cell_end:
2229 }
2230 \int_gincr:N \c@jCol
2231 \@@_rec_preamble_after_col:n
2232 }

```

For ! and @

```

2233 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2234 {
2235 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2236 \@@_rec_preamble:n
2237 }
2238 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2239 \cs_new_protected:cpn { @@ _ | } #1
2240 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2241 \int_incr:N \l_tmpa_int
2242 \@@_make_preamble_i_i:n
2243 }

2244 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2245 {
2246 \str_if_eq:nnTF { #1 } { | }
2247 { \use:c { @@ _ | } | }
2248 { \@@_make_preamble_i_ii:nn { } #1 }
2249 }

2250 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2251 {
2252 \str_if_eq:nnTF { #2 } { [ ]
2253 { \@@_make_preamble_i_iii:nn { #1 } [ ]
2254 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }

```

```

2255 }
2256 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2257 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2258 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2259 {
2260   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2261   \tl_gput_right:Ne \g_@@_array_preamble_tl
2262   {

```

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

```

2263     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2264   }
2265   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2266   {
2267     \@@_vline:n
2268     {
2269       position = \int_eval:n { \c@jCol + 1 } ,
2270       multiplicity = \int_use:N \l_tmpa_int ,
2271       total-width = \dim_use:N \l_@@_rule_width_dim ,
2272       #2
2273     }

```

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.

```

2274   }
2275   \int_zero:N \l_tmpa_int
2276   \str_if_eq:nnF { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2277   \@@_rec_preamble:n #1
2278 }

2279 \cs_new_protected:cpn { @@ _ > } #1 #2
2280 {
2281   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2282   \@@_rec_preamble:n
2283 }
2284 \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.

```

2285 \keys_define:nn { nicematrix / p-column }
2286 {
2287   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2288   r .value_forbidden:n = true ,
2289   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2290   c .value_forbidden:n = true ,
2291   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2292   l .value_forbidden:n = true ,
2293   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2294   S .value_forbidden:n = true ,
2295   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2296   p .value_forbidden:n = true ,
2297   t .meta:n = p ,
2298   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2299   m .value_forbidden:n = true ,
2300   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2301   b .value_forbidden:n = true
2302 }

```

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

```

2303 \cs_new_protected:Npn \@@_p #1
2304 {
2305   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2306     \@@_make_preamble_ii_i:n
2307   }
2308 \cs_set_eq:NN \@@_b \@@_p
2309 \cs_set_eq:NN \@@_m \@@_p
2310 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2311 {
2312   \str_if_eq:nnTF { #1 } { [ ]
2313     { \@@_make_preamble_ii_ii:w [ ]
2314       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2315     }
2316 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2317 { \@@_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.

```

2318 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2319 {

```

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).

```

2320   \str_set:Nn \l_@@_hpos_col_str { j }
2321   \@@_keys_p_column:n { #1 }
2322   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2323 }
2324 \cs_new_protected:Npn \@@_keys_p_column:n #1
2325 { \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.

```

2326 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2327 {
2328   \use:e
2329   {
2330     \@@_make_preamble_ii_v:nnnnnnnn
2331     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2332     { \dim_eval:n { #1 } }
2333   }

```

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.

```

2334     \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2335     { \t1_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2336   }

```

Here, we use `\cs_set_nopar:Npn` instead of `\t1_set:Nn` for efficiency only.

```

2337     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2338     { \str_lowercase:o \l_@@_hpos_col_str }
2339   }
2340 \IfPackageLoadedTF { ragged2e }
2341 {
2342   \str_case:on \l_@@_hpos_col_str
2343   {
2344     c { \exp_not:N \Centering }
2345     l { \exp_not:N \RaggedRight }
2346     r { \exp_not:N \RaggedLeft }
2347   }
2348 }
2349 {
2350   \str_case:on \l_@@_hpos_col_str

```

```

2351     {
2352         c { \exp_not:N \centering }
2353         l { \exp_not:N \raggedright }
2354         r { \exp_not:N \raggedleft }
2355     }
2356 }
2357 #3
2358 }
2359 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2360 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2361 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2362 { #2 }
2363 {
2364     \str_case:onF \l_@@_hpos_col_str
2365     {
2366         { j } { c }
2367         { si } { c }
2368     }

```

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

```

2369     { \str_lowercase:o \l_@@_hpos_col_str }
2370 }
2371 }

```

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

```

2372     \int_gincr:N \c@jCol
2373     \@@_rec_preamble_after_col:n
2374 }

```

#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*.

```

2375 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2376 {
2377     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2378     {
2379         \tl_gput_right:Nn \g_@@_array_preamble_tl
2380         { > \@@_test_if_empty_for_S: }
2381     }
2382     {
2383         \tl_gput_right:Nn \g_@@_array_preamble_tl
2384         { > \@@_test_if_empty: }
2385     }
2386     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2387     \tl_gclear:N \g_@@_pre_cell_tl
2388     \tl_gput_right:Nn \g_@@_array_preamble_tl
2389     {
2390         > {

```

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.

```

2391         \dim_set:Nn \l_@@_col_width_dim { #2 }
2392         \bool_if:NT \c_@@_testphase_table_bool

```

```

2393         { \tag_struct_begin:n { tag = Div } }
2394     \@@_cell_begin:

```

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

```

2395     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2396     \everypar
2397     {
2398         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2399         \everypar { }
2400     }
2401     \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.).

```

2402     #3

```

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

```

2403     \g_@@_row_style_tl
2404     \arraybackslash
2405     #5
2406     }
2407     #8
2408     < {
2409     #6

```

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

```

2410     \@finalstrut \@arstrutbox
2411     \use:c { end #7 }

```

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

```

2412     #4
2413     \@@_cell_end:
2414     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2415     }
2416     }
2417     }

```

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

```

2418 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2419 {

```

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

```

2420 \group_align_safe_begin:
2421 \peek_meaning:NTF &
2422 {
2423     \group_align_safe_end:
2424     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2425     {

```

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

```

2426     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2427     \skip_horizontal:N \l_@@_col_width_dim
2428     }
2429     }
2430     { \group_align_safe_end: }
2431     }

```

```

2432 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2433 {
2434   \peek_meaning:NT \__siunitx_table_skip:n
2435   { \bool_gset_true:N \g_@@_empty_cell_bool }
2436 }

```

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.

```

2437 \cs_new_protected:Npn \@@_center_cell_box:
2438 {

```

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

```

2439   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2440   {
2441     \int_compare:nNnT
2442     { \box_ht:N \l_@@_cell_box }
2443     >

```

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).

```

2444     { \box_ht:N \strutbox }
2445     {
2446       \hbox_set:Nn \l_@@_cell_box
2447       {
2448         \box_move_down:nn
2449         {
2450           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2451             + \baselineskip ) / 2
2452         }
2453         { \box_use:N \l_@@_cell_box }
2454       }
2455     }
2456   }
2457 }

```

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

```

2458 \cs_new_protected:Npn \@@_V #1 #2
2459 {
2460   \str_if_eq:nnTF { #1 } { [ ]
2461     { \@@_make_preamble_V_i:w [ ]
2462       { \@@_make_preamble_V_i:w [ ] { #2 } }
2463     }
2464   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2465     { \@@_make_preamble_V_ii:nn { #1 } }
2466   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2467   {
2468     \str_set:Nn \l_@@_vpos_col_str { p }
2469     \str_set:Nn \l_@@_hpos_col_str { j }
2470     \@@_keys_p_column:n { #1 }
2471     \IfPackageLoadedTF { varwidth }
2472     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2473     {
2474       \@@_error_or_warning:n { varwidth-not-loaded }
2475       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2476     }
2477   }

```

For w and W

```
2478 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2479 \cs_new_protected:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }
```

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

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

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

#4 is the width of the column.

```
2480 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2481 {
2482   \str_if_eq:nnTF { #3 } { s }
2483   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2484   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2485 }
```

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.

```
2486 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2487 {
2488   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2489   \tl_gclear:N \g_@@_pre_cell_tl
2490   \tl_gput_right:Nn \g_@@_array_preamble_tl
2491   {
2492     > {
2493       \dim_set:Nn \l_@@_col_width_dim { #2 }
2494       \@@_cell_begin:
2495       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2496     }
2497     c
2498     < {
2499       \@@_cell_end_for_w_s:
2500       #1
2501       \@@_adjust_size_box:
2502       \box_use_drop:N \l_@@_cell_box
2503     }
2504   }
2505   \int_gincr:N \c@jCol
2506   \@@_rec_preamble_after_col:n
2507 }
```

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

```
2508 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2509 {
2510   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2511   \tl_gclear:N \g_@@_pre_cell_tl
2512   \tl_gput_right:Nn \g_@@_array_preamble_tl
2513   {
2514     > {
```

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.

```
2515       \dim_set:Nn \l_@@_col_width_dim { #4 }
2516       \hbox_set:Nw \l_@@_cell_box
2517       \@@_cell_begin:
2518       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2519     }
2520     c
2521     < {
2522       \@@_cell_end:
2523       \hbox_set_end:
2524       #1
```

```

2525         \@@_adjust_size_box:
2526         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2527     }
2528 }

```

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

```

2529     \int_gincr:N \c@jCol
2530     \@@_rec_preamble_after_col:n
2531 }

2532 \cs_new_protected:Npn \@@_special_W:
2533 {
2534     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2535     { \@@_warning:n { W~warning } }
2536 }

```

For S (of siunitx).

```

2537 \cs_new_protected:Npn \@@_S #1 #2
2538 {
2539     \str_if_eq:nnTF { #2 } { [ ]
2540         { \@@_make_preamble_S:w [ ] }
2541         { \@@_make_preamble_S:w [ ] { #2 } }
2542     }

2543     \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2544     { \@@_make_preamble_S_i:n { #1 } }

2545     \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2546     {
2547         \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2548         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2549         \tl_gclear:N \g_@@_pre_cell_tl
2550         \tl_gput_right:Nn \g_@@_array_preamble_tl
2551         {
2552             > {
2553                 \@@_cell_begin:
2554                 \keys_set:nn { siunitx } { #1 }
2555                 \siunitx_cell_begin:w
2556             }
2557             c
2558             < { \siunitx_cell_end: \@@_cell_end: }
2559         }

```

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

```

2560     \int_gincr:N \c@jCol
2561     \@@_rec_preamble_after_col:n
2562 }

```

For (, [and \{.

```

2563 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2564 {
2565     \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.

```

2566     \int_if_zero:nTF \c@jCol
2567     {
2568         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2569         {

```

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

```

2570         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2571         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2572         \@@_rec_preamble:n #2
2573     }

```

```

2574     {
2575         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2576         \@@_make_preamble_iv:nn { #1 } { #2 }
2577     }
2578 }
2579 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2580 }
2581 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2582 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2583 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2584 {
2585     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2586     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2587     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2588     {
2589         \@@_error:nn { delimiter~after~opening } { #2 }
2590         \@@_rec_preamble:n
2591     }
2592     { \@@_rec_preamble:n #2 }
2593 }

```

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

```

2594 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2595 { \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}`).

```

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

```

```

2629     {
2630     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2631     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2632     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2633     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2634     }
2635     {
2636     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2637     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2638     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2639     \@@_error:nn { double~closing~delimiter } { #2 }
2640     }
2641   }
2642   {
2643   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2644   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2645   \@@_error:nn { double~closing~delimiter } { #2 }
2646   \@@_rec_preamble:n #3
2647   }
2648 }

2649 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2650 { \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 @{...}.

```

2651 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2652 {
2653   \str_if_eq:nnTF { #1 } { < }
2654   \@@_rec_preamble_after_col_i:n
2655   {
2656     \str_if_eq:nnTF { #1 } { @ }
2657     \@@_rec_preamble_after_col_ii:n
2658     {
2659       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2660       {
2661         \tl_gput_right:Nn \g_@@_array_preamble_tl
2662         { ! { \skip_horizontal:N \arrayrulewidth } }
2663       }
2664       {
2665         \clist_if_in:NeT \l_@@_vlines_clist
2666         { \int_eval:n { \c@jCol + 1 } }
2667         {
2668           \tl_gput_right:Nn \g_@@_array_preamble_tl
2669           { ! { \skip_horizontal:N \arrayrulewidth } }
2670         }
2671       }
2672     }
2673   }
2674 }
2675 }

2676 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2677 {
2678   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2679   \@@_rec_preamble_after_col:n
2680 }

```

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.

```

2681 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2682 {

```

```

2683 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2684 {
2685   \tl_gput_right:Nn \g_@@_array_preamble_tl
2686   { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2687 }
2688 {
2689   \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2690   {
2691     \tl_gput_right:Nn \g_@@_array_preamble_tl
2692     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2693   }
2694   { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2695 }
2696 \@@_rec_preamble:n
2697 }

2698 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2699 {
2700   \tl_clear:N \l_tmpa_tl
2701   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2702   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2703 }

```

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.

```

2704 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }

```

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

```

2705 \cs_new_protected:Npn \@@_X #1 #2
2706 {
2707   \str_if_eq:nnTF { #2 } { [ ]
2708     { \@@_make_preamble_X:w [ ] }
2709     { \@@_make_preamble_X:w [ ] #2 }
2710 }
2711 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2712 { \@@_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`).

```

2713 \keys_define:nn { nicematrix / X-column }
2714 { 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.

```

2715 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2716 {

```

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).

```

2717   \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).

```

2718   \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`.

```
2719 \int_zero_new:N \l_@@_weight_int
2720 \int_set_eq:NN \l_@@_weight_int \c_one_int
2721 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2722 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2723 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2724 {
2725   \@@_error_or_warning:n { negative-weight }
2726   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2727 }
2728 \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).

```
2729 \bool_if:NTF \l_@@_X_columns_aux_bool
2730 {
2731   \@@_make_preamble_ii_iv:nnn
2732   { \l_@@_weight_int \l_@@_X_columns_dim }
2733   { minipage }
2734   { \@@_no_update_width: }
2735 }
2736 {
2737   \tl_gput_right:Nn \g_@@_array_preamble_tl
2738   {
2739     > {
2740       \@@_cell_begin:
2741       \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`.

```
2742 \NotEmpty
```

The following code will nullify the box of the cell.

```
2743 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2744 { \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`.

```
2745 \begin { minipage } { 5 cm } \arraybackslash
2746 }
2747 c
2748 < {
2749   \end { minipage }
2750   \@@_cell_end:
2751 }
2752 }
2753 \int_gincr:N \c@jCol
2754 \@@_rec_preamble_after_col:n
2755 }
2756 }

2757 \cs_new_protected:Npn \@@_no_update_width:
2758 {
2759   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2760   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2761 }
```

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

```

2762 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2763 {
2764   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2765     { \int_eval:n { \c@jCol + 1 } }
2766   \tl_gput_right:Ne \g_@@_array_preamble_tl
2767     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2768   \@@_rec_preamble:n
2769 }

```

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.

```

2770 \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).

```

2771 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2772 { \@@_fatal:n { Preamble-forgotten } }
2773 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2774 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2775 \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).

```

2776 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2777 {

```

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`.

```

2778   \multispan { #1 }
2779   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2780   \begingroup
2781   \bool_if:NT \c_@@_testphase_table_bool
2782     { \tbl_update_multicolumn_cell_data:n { #1 } }
2783   \cs_set_nopar:Npn \@addamp
2784     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2785   \tl_gclear:N \g_@@_preamble_tl
2786   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2787   \exp_args:No \@mkpream \g_@@_preamble_tl
2788   \@addtopreamble \@empty
2789   \endgroup
2790   \bool_if:NT \c_@@_recent_array_bool
2791     { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2792   \int_compare:nNnT { #1 } > \c_one_int
2793     {
2794       \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2795         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2796       \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }

```

```

2797 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2798 {
2799   {
2800     \int_if_zero:nTF \c@jCol
2801       { \int_eval:n { \c@iRow + 1 } }
2802       { \int_use:N \c@iRow }
2803   }
2804   { \int_eval:n { \c@jCol + 1 } }
2805   {
2806     \int_if_zero:nTF \c@jCol
2807       { \int_eval:n { \c@iRow + 1 } }
2808       { \int_use:N \c@iRow }
2809   }
2810   { \int_eval:n { \c@jCol + #1 } }
2811   { } % for the name of the block
2812 }
2813 }

```

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

```

2814 \RenewDocumentCommand \cellcolor { 0 { } m }
2815 {
2816   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2817   {
2818     \@@_rectanglecolor [ ##1 ]
2819     { \exp_not:n { ##2 } }
2820     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2821     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2822   }
2823   \ignorespaces
2824 }

```

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

```

2825 \cs_set_nopar:Npn \@sharp { #3 }
2826 \@arstrut
2827 \@preamble
2828 \null

```

We add some lines.

```

2829 \int_gadd:Nn \c@jCol { #1 - 1 }
2830 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2831   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2832 \ignorespaces
2833 }

```

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`.

```

2834 \cs_new_protected:Npn \@_make_m_preamble:n #1
2835 {
2836   \str_case:nnF { #1 }
2837   {
2838     c { \@_make_m_preamble_i:n #1 }
2839     l { \@_make_m_preamble_i:n #1 }
2840     r { \@_make_m_preamble_i:n #1 }
2841     > { \@_make_m_preamble_ii:nn #1 }
2842     ! { \@_make_m_preamble_ii:nn #1 }
2843     @ { \@_make_m_preamble_ii:nn #1 }
2844     | { \@_make_m_preamble_iii:n #1 }
2845     p { \@_make_m_preamble_iv:nnn t #1 }
2846     m { \@_make_m_preamble_iv:nnn c #1 }
2847     b { \@_make_m_preamble_iv:nnn b #1 }
2848     w { \@_make_m_preamble_v:nnnn { } #1 }

```

```

2849     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2850     \q_stop { }
2851   }
2852   {
2853     \cs_if_exist:cTF { NC @ find @ #1 }
2854     {
2855       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2856       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2857     }
2858     {
2859       \str_if_eq:nnTF { #1 } { S }
2860       { \@@_fatal:n { unknown~column~type~S } }
2861       { \@@_fatal:nn { unknown~column~type } { #1 } }
2862     }
2863   }
2864 }

```

For c, l and r

```

2865 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2866 {
2867   \tl_gput_right:Nn \g_@@_preamble_tl
2868   {
2869     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2870     #1
2871     < \@@_cell_end:
2872   }

```

We test for the presence of a <.

```

2873   \@@_make_m_preamble_x:n
2874 }

```

For >, ! and @

```

2875 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2876 {
2877   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2878   \@@_make_m_preamble:n
2879 }

```

For |

```

2880 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2881 {
2882   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2883   \@@_make_m_preamble:n
2884 }

```

For p, m and b

```

2885 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2886 {
2887   \tl_gput_right:Nn \g_@@_preamble_tl
2888   {
2889     > {
2890       \@@_cell_begin:
2891       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2892       \mode_leave_vertical:
2893       \arraybackslash
2894       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2895     }
2896     c
2897     < {
2898       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2899       \end { minipage }
2900       \@@_cell_end:
2901     }
2902   }

```

We test for the presence of a <.

```
2903 \@@_make_m_preamble_x:n
2904 }
```

For w and W

```
2905 \cs_new_protected:Npn \@@_make_m_preamble_v:nmmn #1 #2 #3 #4
2906 {
2907 \tl_gput_right:Nn \g_@@_preamble_tl
2908 {
2909 > {
2910 \dim_set:Nn \l_@@_col_width_dim { #4 }
2911 \hbox_set:Nw \l_@@_cell_box
2912 \@@_cell_begin:
2913 \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2914 }
2915 c
2916 < {
2917 \@@_cell_end:
2918 \hbox_set_end:
2919 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2920 #1
2921 \@@_adjust_size_box:
2922 \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2923 }
2924 }
```

We test for the presence of a <.

```
2925 \@@_make_m_preamble_x:n
2926 }
```

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

```
2927 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2928 {
2929 \str_if_eq:nnTF { #1 } { < }
2930 \@@_make_m_preamble_ix:n
2931 { \@@_make_m_preamble:n { #1 } }
2932 }
2933 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2934 {
2935 \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2936 \@@_make_m_preamble_x:n
2937 }
```

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`).

```
2938 \cs_new_protected:Npn \@@_put_box_in_flow:
2939 {
2940 \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2941 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2942 \str_if_eq:eeTF \l_@@_baseline_tl { c }
2943 { \box_use_drop:N \l_tmpa_box }
2944 \@@_put_box_in_flow_i:
2945 }
```

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

```
2946 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2947 {
2948 \pgfpicture
```

```

2949 \@@_qpoint:n { row - 1 }
2950 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2951 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2952 \dim_gadd:Nn \g_tmpa_dim \pgf@y
2953 \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).

```

2954 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2955 {
2956   \int_set:Nn \l_tmpa_int
2957   {
2958     \str_range:Nnn
2959     \l_@@_baseline_tl
2960     6
2961     { \tl_count:o \l_@@_baseline_tl }
2962   }
2963   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2964 }
2965 {
2966   \str_if_eq:eeTF \l_@@_baseline_tl { t }
2967   { \int_set_eq:NN \l_tmpa_int \c_one_int }
2968   {
2969     \str_if_eq:onTF \l_@@_baseline_tl { b }
2970     { \int_set_eq:NN \l_tmpa_int \c@iRow }
2971     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2972   }
2973   \bool_lazy_or:nnT
2974   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2975   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2976   {
2977     \@@_error:n { bad-value-for-baseline }
2978     \int_set_eq:NN \l_tmpa_int \c_one_int
2979   }
2980   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2981 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2982 }
2983 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2984 \endpgfpicture
2985 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2986 \box_use_drop:N \l_tmpa_box
2987 }

```

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).

```

2988 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2989 {

```

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.

```

2990 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2991 {
2992   \int_compare:nNnT \c@jCol > \c_one_int
2993   {
2994     \box_set_wd:Nn \l_@@_the_array_box
2995     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2996   }
2997 }

```

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

```

2998   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2999   \bool_if:NT \l_@@_caption_above_bool
3000   {
3001     \tl_if_empty:NF \l_@@_caption_tl
3002     {
3003       \bool_set_false:N \g_@@_caption_finished_bool
3004       \int_gzero:N \c@tabularnote
3005       \@@_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).

```

3006       \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3007       {
3008         \tl_gput_right:Ne \g_@@_aux_tl
3009         {
3010           \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3011             { \int_use:N \g_@@_notes_caption_int }
3012         }
3013         \int_gzero:N \g_@@_notes_caption_int
3014       }
3015     }
3016   }

```

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

```

3017   \hbox
3018   {
3019     \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.

```

3020     \@@_create_extra_nodes:
3021     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3022   }

```

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).

```

3023   \bool_lazy_any:nT
3024   {
3025     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3026     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3027     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3028   }
3029   \@@_insert_tabularnotes:
3030   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3031   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3032   \end { minipage }
3033 }

```

```

3034 \cs_new_protected:Npn \@@_insert_caption:
3035 {
3036   \tl_if_empty:NF \l_@@_caption_tl
3037   {
3038     \cs_if_exist:NTF \@capttype
3039     { \@@_insert_caption_i: }
3040     { \@@_error:n { caption~outside~float } }
3041   }
3042 }

```

```

3043 \cs_new_protected:Npn \@@_insert_caption_i:
3044 {
3045   \group_begin:

```

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

```

3046   \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.

```

3047   \IfPackageLoadedT { floatrow }
3048     { \cs_set_eq:NN \@makecaption \FR@makecaption }
3049   \tl_if_empty:NTF \l_@@_short_caption_tl
3050     { \caption }
3051     { \caption [ \l_@@_short_caption_tl ] }
3052     { \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.

```

3053   \bool_if:NF \g_@@_caption_finished_bool
3054     {
3055       \bool_gset_true:N \g_@@_caption_finished_bool
3056       \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3057       \int_gzero:N \c@tabularnote
3058     }
3059   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3060   \group_end:
3061 }

```

```

3062 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3063 {
3064   \@@_error_or_warning:n { tabularnote-below-the-tabular }
3065   \@@_gredirect_none:n { tabularnote~below-the~tabular }
3066 }

```

```

3067 \cs_new_protected:Npn \@@_insert_tabularnotes:
3068 {
3069   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3070   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3071   \skip_vertical:N 0.65ex

```

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

```

3072   \group_begin:
3073   \l_@@_notes_code_before_tl
3074   \tl_if_empty:NF \g_@@_tabularnote_tl
3075     {
3076       \g_@@_tabularnote_tl \par
3077       \tl_gclear:N \g_@@_tabularnote_tl
3078     }

```

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.

```

3079   \int_compare:nNnT \c@tabularnote > \c_zero_int
3080     {
3081       \bool_if:NTF \l_@@_notes_para_bool
3082         {
3083           \begin { tabularnotes* }
3084             \seq_map_inline:Nn \g_@@_notes_seq
3085               { \@@_one_tabularnote:nn ##1 }
3086             \strut
3087           \end { tabularnotes* }

```

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

```

3088     \par
3089     }
3090     {
3091     \tabularnotes
3092     \seq_map_inline:Nn \g_@@_notes_seq
3093     { \@@_one_tabularnote:nn ##1 }
3094     \strut
3095     \endtabularnotes
3096     }
3097 }
3098 \unskip
3099 \group_end:
3100 \bool_if:NT \l_@@_notes_bottomrule_bool
3101 {
3102     \IfPackageLoadedTF { booktabs }
3103     {

```

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

```

3104     \skip_vertical:N \aboverulesep

```

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

```

3105     { \CT@arc@ \hrule height \heavyrulewidth }
3106     }
3107     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3108 }
3109 \l_@@_notes_code_after_tl
3110 \seq_gclear:N \g_@@_notes_seq
3111 \seq_gclear:N \g_@@_notes_in_caption_seq
3112 \int_gzero:N \c@tabularnote
3113 }

```

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.

```

3114 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3115 {
3116     \tl_if_novalue:nTF { #1 }
3117     { \item }
3118     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3119 }

```

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`.

```

3120 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3121 {
3122     \pgfpicture
3123     \@@_qpoint:n { row - 1 }
3124     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3125     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3126     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3127     \endpgfpicture
3128     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3129     \int_if_zero:nT \l_@@_first_row_int
3130     {
3131         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3132         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3133     }
3134     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3135 }

```

Now, the general case.

```
3136 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3137 {
```

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

```
3138 \str_if_eq:eeT \l_@@_baseline_tl { t }
3139 { \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`.

```
3140 \pgfpicture
3141 \@@_qpoint:n { row - 1 }
3142 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3143 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3144 {
3145 \int_set:Nn \l_tmpa_int
3146 {
3147 \str_range:Nnn
3148 \l_@@_baseline_tl
3149 6
3150 { \tl_count:o \l_@@_baseline_tl }
3151 }
3152 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3153 }
3154 {
3155 \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3156 \bool_lazy_or:nnT
3157 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3158 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3159 {
3160 \@@_error:n { bad-value-for-baseline }
3161 \int_set:Nn \l_tmpa_int 1
3162 }
3163 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3164 }
3165 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3166 \endpgfpicture
3167 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3168 \int_if_zero:nT \l_@@_first_row_int
3169 {
3170 \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3171 \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3172 }
3173 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3174 }
```

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.

```
3175 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3176 {
```

We will compute the real width of both delimiters used.

```
3177 \dim_zero_new:N \l_@@_real_left_delim_dim
3178 \dim_zero_new:N \l_@@_real_right_delim_dim
3179 \hbox_set:Nn \l_tmpb_box
3180 {
3181 \m@th % added 2024/11/21
3182 \c_math_toggle_token
3183 \left #1
3184 \vcenter
3185 {
3186 \vbox_to_ht:nn
3187 { \box_ht_plus_dp:N \l_tmpa_box }
```

```

3188         { }
3189     }
3190     \right .
3191     \c_math_toggle_token
3192 }
3193 \dim_set:Nn \l_@@_real_left_delim_dim
3194 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3195 \hbox_set:Nn \l_tmpb_box
3196 {
3197     \m@th % added 2024/11/21
3198     \c_math_toggle_token
3199     \left .
3200     \vbox_to_ht:nn
3201     { \box_ht_plus_dp:N \l_tmpa_box }
3202     { }
3203     \right #2
3204     \c_math_toggle_token
3205 }
3206 \dim_set:Nn \l_@@_real_right_delim_dim
3207 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3208     \skip_horizontal:N \l_@@_left_delim_dim
3209     \skip_horizontal:N -\l_@@_real_left_delim_dim
3210     \@@_put_box_in_flow:
3211     \skip_horizontal:N \l_@@_right_delim_dim
3212     \skip_horizontal:N -\l_@@_real_right_delim_dim
3213 }

```

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).

```

3214 \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).

```

3215 {
3216     \peek_remove_spaces:n
3217     {
3218         \peek_meaning:NTF \end
3219         \@@_analyze_end:Nn
3220         {
3221             \@@_transform_preamble:

```

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

```

3222         \@@_array:o \g_@@_array_preamble_tl
3223     }
3224 }
3225 }
3226 {
3227     \@@_create_col_nodes:
3228     \endarray
3229 }

```

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

```

3230 \NewDocumentEnvironment { @@-light-syntax } { b }
3231 {

```

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.

```

3232   \tl_if_empty:nT { #1 }
3233     { \@@_fatal:n { empty~environment } }
3234   \tl_if_in:nnT { #1 } { & }
3235     { \@@_fatal:n { ampersand~in~light~syntax } }
3236   \tl_if_in:nnT { #1 } { \ }
3237     { \@@_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`.

```

3238   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3239   }

```

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.

```

3240   {
3241     \@@_create_col_nodes:
3242     \endarray
3243   }
3244   \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3245     {
3246       \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).

```

3247     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3248     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3249     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3250       \seq_set_split:Nee
3251       \seq_set_split:Nnon
3252       \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3253     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3254     \tl_if_empty:NF \l_tmpa_tl
3255     { \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.

```

3256     \int_compare:nNnT \l_@@_last_row_int = { -1 }
3257     { \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`).

```

3258     \tl_build_begin:N \l_@@_new_body_tl
3259     \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3260     \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3261     \@@_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).

```

3262   \seq_map_inline:Nn \l_@@_rows_seq
3263   {
3264     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3265     \@@_line_with_light_syntax:n { #1 }
3266   }
3267   \tl_build_end:N \l_@@_new_body_tl
3268   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3269   {
3270     \int_set:Nn \l_@@_last_col_int
3271     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3272   }

```

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.

```

3273   \@@_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`).

```

3274   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3275   }
3276   \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3277   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3278   {
3279     \seq_clear_new:N \l_@@_cells_seq
3280     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3281     \int_set:Nn \l_@@_nb_cols_int
3282     {
3283       \int_max:nn
3284       \l_@@_nb_cols_int
3285       { \seq_count:N \l_@@_cells_seq }
3286     }
3287     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3288     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3289     \seq_map_inline:Nn \l_@@_cells_seq
3290     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3291   }

```

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`.

```

3292   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3293   {
3294     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3295     { \@@_fatal:n { empty-environment } }

```

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

```

3296     \end { #2 }
3297   }

```

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`).

```

3298   \cs_new:Npn \@@_create_col_nodes:
3299   {
3300     \crrc
3301     \int_if_zero:nT \l_@@_first_col_int
3302     {
3303       \omit

```

```

3304     \hbox_overlap_left:n
3305     {
3306         \bool_if:NT \l_@@_code_before_bool
3307         { \pgfsys@markposition { \@@_env: - col - 0 } }
3308         \pgfpicture
3309         \pgfrememberpicturepositiononpagetrue
3310         \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3311         \str_if_empty:NF \l_@@_name_str
3312         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3313         \endpgfpicture
3314         \skip_horizontal:N 2\col@sep
3315         \skip_horizontal:N \g_@@_width_first_col_dim
3316     }
3317     &
3318 }
3319 \omit

```

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

```

3320     \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`).

```

3321     \int_if_zero:nTF \l_@@_first_col_int
3322     {
3323         \bool_if:NT \l_@@_code_before_bool
3324         {
3325             \hbox
3326             {
3327                 \skip_horizontal:N -0.5\arrayrulewidth
3328                 \pgfsys@markposition { \@@_env: - col - 1 }
3329                 \skip_horizontal:N 0.5\arrayrulewidth
3330             }
3331         }
3332         \pgfpicture
3333         \pgfrememberpicturepositiononpagetrue
3334         \pgfcoordinate { \@@_env: - col - 1 }
3335         { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3336         \str_if_empty:NF \l_@@_name_str
3337         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3338         \endpgfpicture
3339     }
3340     {
3341         \bool_if:NT \l_@@_code_before_bool
3342         {
3343             \hbox
3344             {
3345                 \skip_horizontal:N 0.5\arrayrulewidth
3346                 \pgfsys@markposition { \@@_env: - col - 1 }
3347                 \skip_horizontal:N -0.5\arrayrulewidth
3348             }
3349         }
3350         \pgfpicture
3351         \pgfrememberpicturepositiononpagetrue
3352         \pgfcoordinate { \@@_env: - col - 1 }
3353         { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3354         \str_if_empty:NF \l_@@_name_str
3355         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3356         \endpgfpicture
3357     }

```

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.

```

3358 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3359 \bool_if:NF \l_@@_auto_columns_width_bool
3360 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3361 {
3362   \bool_lazy_and:nnTF
3363     \l_@@_auto_columns_width_bool
3364     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3365     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3366     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3367   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3368 }
3369 \skip_horizontal:N \g_tmpa_skip
3370 \hbox
3371 {
3372   \bool_if:NT \l_@@_code_before_bool
3373   {
3374     \hbox
3375     {
3376       \skip_horizontal:N -0.5\arrayrulewidth
3377       \pgfsys@markposition { \@@_env: - col - 2 }
3378       \skip_horizontal:N 0.5\arrayrulewidth
3379     }
3380   }
3381   \pgfpicture
3382   \pgfrememberpicturepositiononpagetrue
3383   \pgfcoordinate { \@@_env: - col - 2 }
3384   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3385   \str_if_empty:NF \l_@@_name_str
3386   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3387   \endpgfpicture
3388 }

```

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.

```

3389 \int_gset_eq:NN \g_tmpa_int \c_one_int
3390 \bool_if:NTF \g_@@_last_col_found_bool
3391 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3392 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3393 {
3394   &
3395   \omit
3396   \int_gincr:N \g_tmpa_int

```

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

```

3397   \skip_horizontal:N \g_tmpa_skip
3398   \bool_if:NT \l_@@_code_before_bool
3399   {
3400     \hbox
3401     {
3402       \skip_horizontal:N -0.5\arrayrulewidth
3403       \pgfsys@markposition
3404       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3405       \skip_horizontal:N 0.5\arrayrulewidth
3406     }
3407   }

```

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

```

3408   \pgfpicture
3409   \pgfrememberpicturepositiononpagetrue
3410   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3411   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3412   \str_if_empty:NF \l_@@_name_str

```

```

3413     {
3414         \pgfnodealias
3415         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3416         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3417     }
3418 \endpgfpicture
3419 }

3420 &
3421 \omit

```

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

```

3422 \int_if_zero:nT \g_@@_col_total_int
3423   { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3424 \skip_horizontal:N \g_tmpa_skip
3425 \int_gincr:N \g_tmpa_int
3426 \bool_lazy_any:nF
3427   {
3428     \g_@@_delims_bool
3429     \l_@@_tabular_bool
3430     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3431     \l_@@_exterior_arraycolsep_bool
3432     \l_@@_bar_at_end_of_pream_bool
3433   }
3434   { \skip_horizontal:N -\col@sep }
3435 \bool_if:NT \l_@@_code_before_bool
3436   {
3437     \hbox
3438     {
3439       \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.

```

3440     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3441     { \skip_horizontal:N -\arraycolsep }
3442 \pgfsys@markposition
3443   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3444   \skip_horizontal:N 0.5\arrayrulewidth
3445   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3446   { \skip_horizontal:N \arraycolsep }
3447   }
3448 }
3449 \pgfpicture
3450 \pgfrememberpicturepositiononpagetrue
3451 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3452   {
3453     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3454     {
3455       \pgfpoint
3456       { - 0.5 \arrayrulewidth - \arraycolsep }
3457       \c_zero_dim
3458     }
3459     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3460   }
3461 \str_if_empty:NF \l_@@_name_str
3462   {
3463     \pgfnodealias
3464     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3465     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3466   }
3467 \endpgfpicture

```

```

3468 \bool_if:NT \g_@@_last_col_found_bool
3469 {
3470   \hbox_overlap_right:n
3471   {
3472     \skip_horizontal:N \g_@@_width_last_col_dim
3473     \skip_horizontal:N \col@sep
3474     \bool_if:NT \l_@@_code_before_bool
3475     {
3476       \pgfsys@markposition
3477       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3478     }
3479     \pgfpicture
3480     \pgfrememberpicturepositiononpagetrue
3481     \pgfcoordinate
3482     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3483     \pgfpointorigin
3484     \str_if_empty:NF \l_@@_name_str
3485     {
3486       \pgfnodealias
3487       {
3488         \l_@@_name_str - col
3489         - \int_eval:n { \g_@@_col_total_int + 1 }
3490       }
3491       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3492     }
3493     \endpgfpicture
3494   }
3495 }
3496 % \cr
3497 }

```

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

```

3498 \tl_const:Nn \c_@@_preamble_first_col_tl
3499 {
3500   >
3501   {

```

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

```

3502     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3503     \bool_gset_true:N \g_@@_after_col_zero_bool
3504     \@@_begin_of_row:
3505     \hbox_set:Nw \l_@@_cell_box
3506     \@@_math_toggle:
3507     \@@_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”.

```

3508     \int_compare:nNnT \c@iRow > \c_zero_int
3509     {
3510       \bool_lazy_or:nnT
3511       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3512       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3513       {
3514         \l_@@_code_for_first_col_tl
3515         \xglobal \colorlet { nicematrix-first-col } { . }
3516       }
3517     }
3518 }

```

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`.

```

3519   l

```

```

3520 <
3521 {
3522   \@@_math_toggle:
3523   \hbox_set_end:
3524   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3525   \@@_adjust_size_box:
3526   \@@_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.

```

3527   \dim_gset:Nn \g_@@_width_first_col_dim
3528   { \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.

```

3529   \hbox_overlap_left:n
3530   {
3531     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3532     \@@_node_for_cell:
3533     { \box_use_drop:N \l_@@_cell_box }
3534     \skip_horizontal:N \l_@@_left_delim_dim
3535     \skip_horizontal:N \l_@@_left_margin_dim
3536     \skip_horizontal:N \l_@@_extra_left_margin_dim
3537   }
3538   \bool_gset_false:N \g_@@_empty_cell_bool
3539   \skip_horizontal:N -2\col@sep
3540 }
3541 }

```

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

```

3542 \tl_const:Nn \c_@@_preamble_last_col_tl
3543 {
3544   >
3545   {
3546     \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).

```

3547   \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.

```

3548   \bool_gset_true:N \g_@@_last_col_found_bool
3549   \int_gincr:N \c@jCol
3550   \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3551   \hbox_set:Nw \l_@@_cell_box
3552   \@@_math_toggle:
3553   \@@_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”.

```

3554   \int_compare:nNnT \c@iRow > \c_zero_int
3555   {
3556     \bool_lazy_or:nnT
3557     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3558     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3559     {
3560       \l_@@_code_for_last_col_tl
3561       \xglobal \colorlet { nicematrix-last-col } { . }
3562     }
3563   }
3564 }
3565 1
3566 <
3567 {
3568   \@@_math_toggle:
3569   \hbox_set_end:
3570   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

```

3571 \@@_adjust_size_box:
3572 \@@_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.

```

3573 \dim_gset:Nn \g_@@_width_last_col_dim
3574 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3575 \skip_horizontal:N -2\col@sep

```

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

```

3576 \hbox_overlap_right:n
3577 {
3578   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3579   {
3580     \skip_horizontal:N \l_@@_right_delim_dim
3581     \skip_horizontal:N \l_@@_right_margin_dim
3582     \skip_horizontal:N \l_@@_extra_right_margin_dim
3583     \@@_node_for_cell:
3584   }
3585 }
3586 \bool_gset_false:N \g_@@_empty_cell_bool
3587 }
3588 }

```

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

```

3589 \NewDocumentEnvironment { NiceArray } { }
3590 {
3591   \bool_gset_false:N \g_@@_delims_bool
3592   \str_if_empty:NT \g_@@_name_env_str
3593   { \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).

```

3594   \NiceArrayWithDelims . .
3595 }
3596 { \endNiceArrayWithDelims }

```

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

```

3597 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3598 {
3599   \NewDocumentEnvironment { #1 NiceArray } { }
3600   {
3601     \bool_gset_true:N \g_@@_delims_bool
3602     \str_if_empty:NT \g_@@_name_env_str
3603     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3604     \@@_test_if_math_mode:
3605     \NiceArrayWithDelims #2 #3
3606   }
3607   { \endNiceArrayWithDelims }
3608 }
3609 \@@_def_env:nnn p ( )
3610 \@@_def_env:nnn b [ ]
3611 \@@_def_env:nnn B \{ \}
3612 \@@_def_env:nnn v | |
3613 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3614 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3615 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3616 {
3617   \bool_set_false:N \l_@@_preamble_bool
3618   \tl_clear:N \l_tmpa_tl
3619   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3620     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3621   \tl_put_right:Nn \l_tmpa_tl
3622     {
3623       *
3624       {
3625         \int_case:nnF \l_@@_last_col_int
3626           {
3627             { -2 } { \c@MaxMatrixCols }
3628             { -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).

```

3629       }
3630       { \int_eval:n { \l_@@_last_col_int - 1 } }
3631     }
3632     { #2 }
3633   }
3634   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3635   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3636 }
3637 \clist_map_inline:nn { p , b , B , v , V }
3638 {
3639   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3640   {
3641     \bool_gset_true:N \g_@@_delims_bool
3642     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3643     \int_if_zero:nT \l_@@_last_col_int
3644     {
3645       \bool_set_true:N \l_@@_last_col_without_value_bool
3646       \int_set:Nn \l_@@_last_col_int { -1 }
3647     }
3648     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3649     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3650   }
3651   { \use:c { end #1 NiceArray } }
3652 }

```

We define also an environment `{NiceMatrix}`

```

3653 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3654 {
3655   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3656   \int_if_zero:nT \l_@@_last_col_int
3657   {
3658     \bool_set_true:N \l_@@_last_col_without_value_bool
3659     \int_set:Nn \l_@@_last_col_int { -1 }
3660   }
3661   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3662   \bool_lazy_or:nnT
3663     { \clist_if_empty_p:N \l_@@_vlines_clist }
3664     { \l_@@_except_borders_bool }
3665     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3666   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3667 }
3668 { \endNiceArray }

```

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

```
3669 \cs_new_protected:Npn \@@_NotEmpty:
3670 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3671 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3672 {
```

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

```
3673   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3674     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3675   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3676   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3677   \tl_if_empty:NF \l_@@_short_caption_tl
3678     {
3679     \tl_if_empty:NT \l_@@_caption_tl
3680       {
3681         \@@_error_or_warning:n { short-caption-without-caption }
3682         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3683       }
3684     }
3685   \tl_if_empty:NF \l_@@_label_tl
3686     {
3687     \tl_if_empty:NT \l_@@_caption_tl
3688       { \@@_error_or_warning:n { label-without-caption } }
3689     }
3690   \NewDocumentEnvironment { TabularNote } { b }
3691     {
3692     \bool_if:NTF \l_@@_in_code_after_bool
3693       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3694       {
3695         \tl_if_empty:NF \g_@@_tabularnote_tl
3696           { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3697         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3698       }
3699     }
3700     { }
3701   \@@_settings_for_tabular:
3702   \NiceArray { #2 }
3703 }
3704 { \endNiceArray }
3705 \cs_new_protected:Npn \@@_settings_for_tabular:
3706 {
3707   \bool_set_true:N \l_@@_tabular_bool
3708   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3709   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3710   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3711 }

3712 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3713 {
3714   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3715   \dim_zero_new:N \l_@@_width_dim
3716   \dim_set:Nn \l_@@_width_dim { #1 }
3717   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3718   \@@_settings_for_tabular:
3719   \NiceArray { #3 }
3720 }
3721 {
3722   \endNiceArray
```

```

3723 \int_if_zero:nT \g_@@_total_X_weight_int
3724 { \@@_error:n { NiceTabularX~without~X } }
3725 }

3726 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3727 {
3728 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3729 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3730 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3731 \@@_settings_for_tabular:
3732 \NiceArray { #3 }
3733 }
3734 { \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`).

```

3735 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3736 {
3737 \bool_lazy_all:nT
3738 {
3739 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3740 \l_@@_hvlines_bool
3741 { ! \g_@@_delims_bool }
3742 { ! \l_@@_except_borders_bool }
3743 }
3744 {
3745 \bool_set_true:N \l_@@_except_borders_bool
3746 \clist_if_empty:NF \l_@@_corners_clist
3747 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3748 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3749 {
3750 \@@_stroke_block:nnn
3751 {
3752 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3753 draw = \l_@@_rules_color_tl
3754 }
3755 { 1-1 }
3756 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3757 }
3758 }
3759 }

```

```

3760 \cs_new_protected:Npn \@@_after_array:
3761 {

```

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`.

```

3762 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3763 \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.

```
3764 \bool_if:NT \g_@@_last_col_found_bool
3765 { \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`.

```
3766 \bool_if:NT \l_@@_last_col_without_value_bool
3767 { \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.

```
3768 \bool_if:NT \l_@@_last_row_without_value_bool
3769 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3770 \tl_gput_right:Ne \g_@@_aux_tl
3771 {
3772   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3773   {
3774     \int_use:N \l_@@_first_row_int ,
3775     \int_use:N \c@iRow ,
3776     \int_use:N \g_@@_row_total_int ,
3777     \int_use:N \l_@@_first_col_int ,
3778     \int_use:N \c@jCol ,
3779     \int_use:N \g_@@_col_total_int
3780   }
3781 }
```

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`).

```
3782 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3783 {
3784   \tl_gput_right:Ne \g_@@_aux_tl
3785   {
3786     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3787     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3788   }
3789 }
3790 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3791 {
3792   \tl_gput_right:Ne \g_@@_aux_tl
3793   {
3794     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3795     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3796     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3797     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3798   }
3799 }
```

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

```
3800 \@@_create_diag_nodes:
```

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

```
3801 \pgfpicture
3802 \int_step_inline:nn \c@iRow
3803 {
3804   \pgfnodealias
3805   { \@@_env: - ##1 - last }
3806   { \@@_env: - ##1 - \int_use:N \c@jCol }
3807 }
3808 \int_step_inline:nn \c@jCol
3809 {
3810   \pgfnodealias
```

```

3811     { \@@_env: - last - ##1 }
3812     { \@@_env: - \int_use:N \c@iRow - ##1 }
3813   }
3814   \str_if_empty:NF \l_@@_name_str
3815   {
3816     \int_step_inline:nn \c@iRow
3817     {
3818       \pgfnodealias
3819       { \l_@@_name_str - ##1 - last }
3820       { \@@_env: - ##1 - \int_use:N \c@jCol }
3821     }
3822     \int_step_inline:nn \c@jCol
3823     {
3824       \pgfnodealias
3825       { \l_@@_name_str - last - ##1 }
3826       { \@@_env: - \int_use:N \c@iRow - ##1 }
3827     }
3828   }
3829   \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.

```

3830   \bool_if:NT \l_@@_parallelize_diags_bool
3831   {
3832     \int_gzero_new:N \g_@@_ddots_int
3833     \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.

```

3834     \dim_gzero_new:N \g_@@_delta_x_one_dim
3835     \dim_gzero_new:N \g_@@_delta_y_one_dim
3836     \dim_gzero_new:N \g_@@_delta_x_two_dim
3837     \dim_gzero_new:N \g_@@_delta_y_two_dim
3838   }
3839   \int_zero_new:N \l_@@_initial_i_int
3840   \int_zero_new:N \l_@@_initial_j_int
3841   \int_zero_new:N \l_@@_final_i_int
3842   \int_zero_new:N \l_@@_final_j_int
3843   \bool_set_false:N \l_@@_initial_open_bool
3844   \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.

```

3845   \bool_if:NT \l_@@_small_bool
3846   {
3847     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3848     \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.

```

3849     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3850     { 0.6 \l_@@_xdots_shorten_start_dim }
3851     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3852     { 0.6 \l_@@_xdots_shorten_end_dim }
3853   }

```

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

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

```
3854 \@@_draw_dotted_lines:
```

The following computes the “corners” (made up of empty cells) but if there is no corner to compute, it won’t do anything. The corners are computed in `\l_@@_corners_cells_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```
3855 \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-*}`).

```
3856 \@@_adjust_pos_of_blocks_seq:
3857 \@@_deal_with_rounded_corners:
3858 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3859 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3860 \IfPackageLoadedT { tikz }
3861 {
3862   \tikzset
3863   {
3864     every-picture / .style =
3865     {
3866       overlay ,
3867       remember-picture ,
3868       name-prefix = \@@_env: -
3869     }
3870   }
3871 }
3872 \bool_if:NT \c_@@_recent_array_bool
3873 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3874 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3875 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3876 \cs_set_eq:NN \OverBrace \@@_OverBrace
3877 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3878 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3879 \cs_set_eq:NN \line \@@_line
3880 \g_@@_pre_code_after_tl
3881 \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.

```
3882 \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).

```
3883 \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`).

```
3884 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3885 { \@@_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:`.

```
3886 \bool_set_true:N \l_@@_in_code_after_bool
3887 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3888 \scan_stop:
3889 \tl_gclear:N \g_nicematrix_code_after_tl
3890 \group_end:
```

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

```

3891   \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3892   \tl_if_empty:NF \g_@@_pre_code_before_tl
3893   {
3894     \tl_gput_right:Ne \g_@@_aux_tl
3895     {
3896       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3897       { \exp_not:o \g_@@_pre_code_before_tl }
3898     }
3899     \tl_gclear:N \g_@@_pre_code_before_tl
3900   }
3901   \tl_if_empty:NF \g_nicematrix_code_before_tl
3902   {
3903     \tl_gput_right:Ne \g_@@_aux_tl
3904     {
3905       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3906       { \exp_not:o \g_nicematrix_code_before_tl }
3907     }
3908     \tl_gclear:N \g_nicematrix_code_before_tl
3909   }

3910   \str_gclear:N \g_@@_name_env_str
3911   \@@_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`.

```

3912   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3913   }

```

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`.

```

3914 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3915 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

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

```

3916 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3917 {
3918   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3919   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3920 }

```

The following command must *not* be protected.

```

3921 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3922 {
3923   { #1 }
3924   { #2 }
3925   {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3926     \int_compare:nNnTF { #3 } > { 99 }
3927         { \int_use:N \c@iRow }
3928         { #3 }
3929     }
3930     {
3931         \int_compare:nNnTF { #4 } > { 99 }
3932             { \int_use:N \c@jCol }
3933             { #4 }
3934     }
3935     { #5 }
3936 }

```

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).

```

3937 \hook_gput_code:nnn { begindocument } { . }
3938 {
3939     \cs_new_protected:Npe \@@_draw_dotted_lines:
3940     {
3941         \c_@@_pgfortikzpicture_tl
3942         \@@_draw_dotted_lines_i:
3943         \c_@@_endpgfortikzpicture_tl
3944     }
3945 }

```

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

```

3946 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3947 {
3948     \pgfrememberpicturepositiononpagetrue
3949     \pgf@relevantforpicturesizefalse
3950     \g_@@_HVdotsfor_lines_tl
3951     \g_@@_Vdots_lines_tl
3952     \g_@@_Ddots_lines_tl
3953     \g_@@_Iddots_lines_tl
3954     \g_@@_Cdots_lines_tl
3955     \g_@@_Ldots_lines_tl
3956 }

```

```

3957 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3958 {
3959     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3960     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3961 }

```

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

```

3962 \pgfdeclareshape { @@_diag_node }
3963 {
3964     \savedanchor { \five }
3965     {
3966         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3967         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3968     }
3969     \anchor { 5 } { \five }
3970     \anchor { center } { \pgfpointorigin }
3971     \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3972     \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3973     \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3974     \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3975     \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }

```

```

3976 \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3977 \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3978 \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3979 \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3980 \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3981 }

```

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).

```

3982 \cs_new_protected:Npn \@@_create_diag_nodes:
3983 {
3984 \pgfpicture
3985 \pgfrememberpicturepositiononpagetrue
3986 \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3987 {
3988 \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3989 \dim_set_eq:NN \l_tmpa_dim \pgf@x
3990 \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3991 \dim_set_eq:NN \l_tmpb_dim \pgf@y
3992 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3993 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3994 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3995 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3996 \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.

```

3997 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3998 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3999 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4000 \str_if_empty:NF \l_@@_name_str
4001 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4002 }

```

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

```

4003 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4004 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4005 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4006 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4007 \pgfcoordinate
4008 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4009 \pgfnodealias
4010 { \@@_env: - last }
4011 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4012 \str_if_empty:NF \l_@@_name_str
4013 {
4014 \pgfnodealias
4015 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4016 { \@@_env: - \int_use:N \l_tmpa_int }
4017 \pgfnodealias
4018 { \l_@@_name_str - last }
4019 { \@@_env: - last }
4020 }
4021 \endpgfpicture
4022 }

```



```

4046         \if_int_compare:w #4 = -1
4047         \bool_set_true:N \l_@@_final_open_bool
4048         \fi:
4049     \else:
4050         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4051         \if_int_compare:w #4 = \c_one_int
4052         \bool_set_true:N \l_@@_final_open_bool
4053         \fi:
4054     \fi:
4055 \fi:
4056 \fi:
4057 \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.

```

4058     {

```

We do a step backwards.

```

4059         \int_sub:Nn \l_@@_final_i_int { #3 }
4060         \int_sub:Nn \l_@@_final_j_int { #4 }
4061         \bool_set_true:N \l_@@_stop_loop_bool
4062     }

```

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`.

```

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

```

4084         {
4085             \cs_set_nopar:cpn
4086             {
4087                 @@ _ dotted _
4088                 \int_use:N \l_@@_final_i_int -
4089                 \int_use:N \l_@@_final_j_int
4090             }
4091             { }
4092         }
4093     }
4094 }
4095 }

```

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

```
4096 \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4097 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
```

```
4098 \bool_do_until:Nn \l_@@_stop_loop_bool
```

```
4099 {
4100 \int_sub:Nn \l_@@_initial_i_int { #3 }
4101 \int_sub:Nn \l_@@_initial_j_int { #4 }
4102 \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.

```
4103 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4104 \if_int_compare:w #3 = \c_one_int
4105 \bool_set_true:N \l_@@_initial_open_bool
4106 \else:
```

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

```
4107 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4108 \bool_set_true:N \l_@@_initial_open_bool
4109 \fi:
4110 \fi:
4111 \else:
4112 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4113 \if_int_compare:w #4 = \c_one_int
4114 \bool_set_true:N \l_@@_initial_open_bool
4115 \fi:
4116 \else:
4117 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4118 \if_int_compare:w #4 = -1
4119 \bool_set_true:N \l_@@_initial_open_bool
4120 \fi:
4121 \fi:
4122 \fi:
4123 \fi:
4124 \bool_if:NTF \l_@@_initial_open_bool
4125 {
4126 \int_add:Nn \l_@@_initial_i_int { #3 }
4127 \int_add:Nn \l_@@_initial_j_int { #4 }
4128 \bool_set_true:N \l_@@_stop_loop_bool
4129 }
4130 {
4131 \cs_if_exist:cTF
4132 {
4133 @@ _ dotted _
4134 \int_use:N \l_@@_initial_i_int -
4135 \int_use:N \l_@@_initial_j_int
4136 }
4137 {
4138 \int_add:Nn \l_@@_initial_i_int { #3 }
4139 \int_add:Nn \l_@@_initial_j_int { #4 }
4140 \bool_set_true:N \l_@@_initial_open_bool
4141 \bool_set_true:N \l_@@_stop_loop_bool
4142 }
4143 {
4144 \cs_if_exist:cTF
4145 {
4146 pgf @ sh @ ns @ \@@_env:
4147 - \int_use:N \l_@@_initial_i_int
4148 - \int_use:N \l_@@_initial_j_int
4149 }
```

```

4150         { \bool_set_true:N \l_@@_stop_loop_bool }
4151         {
4152             \cs_set_nopar:cpn
4153             {
4154                 @@ _ dotted _
4155                 \int_use:N \l_@@_initial_i_int -
4156                 \int_use:N \l_@@_initial_j_int
4157             }
4158         { }
4159     }
4160 }
4161 }
4162 }

```

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.

```

4163 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4164 {
4165     { \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`.

```

4166     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4167     { \int_use:N \l_@@_final_i_int }
4168     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4169     { } % for the name of the block
4170 }
4171 }

```

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

```

4172 \cs_new_protected:Npn \@@_open_shorten:
4173 {
4174     \bool_if:NT \l_@@_initial_open_bool
4175     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4176     \bool_if:NT \l_@@_final_open_bool
4177     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4178 }

```

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).

```

4179 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4180 {
4181     \int_set_eq:NN \l_@@_row_min_int \c_one_int
4182     \int_set_eq:NN \l_@@_col_min_int \c_one_int
4183     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4184     \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`.

```

4185 \seq_if_empty:NF \g_@@_submatrix_seq
4186 {
4187     \seq_map_inline:Nn \g_@@_submatrix_seq
4188     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4189 }
4190 }

```

#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.

```
4191 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4192 {
4193   \if_int_compare:w #3 > #1
4194   \else:
4195     \if_int_compare:w #1 > #5
4196     \else:
4197       \if_int_compare:w #4 > #2
4198       \else:
4199         \if_int_compare:w #2 > #6
4200         \else:
4201           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4202           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4203           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4204           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4205         \fi:
4206       \fi:
4207     \fi:
4208   \fi:
4209 }

4210 \cs_new_protected:Npn \@@_set_initial_coords:
4211 {
4212   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4213   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4214 }
4215 \cs_new_protected:Npn \@@_set_final_coords:
4216 {
4217   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4218   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4219 }
4220 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4221 {
4222   \pgfpointanchor
4223   {
4224     \@@_env:
4225     - \int_use:N \l_@@_initial_i_int
4226     - \int_use:N \l_@@_initial_j_int
4227   }
4228   { #1 }
4229   \@@_set_initial_coords:
```

```

4230 }
4231 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4232 {
4233   \pgfpointanchor
4234   {
4235     \@@_env:
4236     - \int_use:N \l_@@_final_i_int
4237     - \int_use:N \l_@@_final_j_int
4238   }
4239   { #1 }
4240   \@@_set_final_coords:
4241 }
4242 \cs_new_protected:Npn \@@_open_x_initial_dim:
4243 {
4244   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4245   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4246   {
4247     \cs_if_exist:cT
4248     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4249     {
4250       \pgfpointanchor
4251       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4252       { west }
4253       \dim_set:Nn \l_@@_x_initial_dim
4254       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4255     }
4256   }

```

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

```

4257   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4258   {
4259     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4260     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4261     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4262   }
4263 }
4264 \cs_new_protected:Npn \@@_open_x_final_dim:
4265 {
4266   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4267   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4268   {
4269     \cs_if_exist:cT
4270     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4271     {
4272       \pgfpointanchor
4273       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4274       { east }
4275       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4276       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4277     }
4278   }

```

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

```

4279   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4280   {
4281     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4282     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4283     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4284   }
4285 }

```

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.

```

4286 \cs_new_protected:Npn \@@_draw_Ldots:nmn #1 #2 #3
4287 {
4288   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4289   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4290   {
4291     \@@_find_extremities_of_line:nmmm { #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.

```

4292     \group_begin:
4293     \@@_open_shorten:
4294     \int_if_zero:nTF { #1 }
4295     { \color { nicematrix-first-row } }
4296     {

```

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.

```

4297         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4298         { \color { nicematrix-last-row } }
4299     }
4300     \keys_set:nn { nicematrix / xdots } { #3 }
4301     \@@_color:o \l_@@_xdots_color_tl
4302     \@@_actually_draw_Ldots:
4303   \group_end:
4304 }
4305 }

```

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`.

```

4306 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4307 {
4308   \bool_if:NTF \l_@@_initial_open_bool
4309   {
4310     \@@_open_x_initial_dim:
4311     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4312     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4313   }
4314   { \@@_set_initial_coords_from_anchor:n { base-east } }
4315   \bool_if:NTF \l_@@_final_open_bool
4316   {
4317     \@@_open_x_final_dim:
4318     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4319     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4320   }
4321   { \@@_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.

```

4322   \bool_lazy_all:nTF
4323   {
4324     \l_@@_initial_open_bool
4325     \l_@@_final_open_bool

```

```

4326     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4327   }
4328   {
4329     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4330     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4331   }

```

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 (?).

```

4332   {
4333     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4334     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4335   }
4336   \@@_draw_line:
4337 }

```

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.

```

4338 \cs_new_protected:Npn \@@_draw_Cdots:nmn #1 #2 #3
4339   {
4340     \@@_adjust_to_submatrix:nm { #1 } { #2 }
4341     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4342     {
4343       \@@_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.

```

4344     \group_begin:
4345       \@@_open_shorten:
4346       \int_if_zero:nTF { #1 }
4347         { \color { nicematrix-first-row } }
4348       {

```

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.

```

4349         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4350         { \color { nicematrix-last-row } }
4351       }
4352     \keys_set:nm { nicematrix / xdots } { #3 }
4353     \@@_color:o \l_@@_xdots_color_tl
4354     \@@_actually_draw_Cdots:
4355   \group_end:
4356 }
4357 }

```

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`.

```

4358 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4359   {
4360     \bool_if:NTF \l_@@_initial_open_bool
4361       { \@@_open_x_initial_dim: }
4362     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4363     \bool_if:NTF \l_@@_final_open_bool

```

```

4364 { \@@_open_x_final_dim: }
4365 { \@@_set_final_coords_from_anchor:n { mid~west } }
4366 \bool_lazy_and:nnTF
4367 \l_@@_initial_open_bool
4368 \l_@@_final_open_bool
4369 {
4370 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4371 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4372 \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4373 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4374 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4375 }
4376 {
4377 \bool_if:NT \l_@@_initial_open_bool
4378 { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4379 \bool_if:NT \l_@@_final_open_bool
4380 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4381 }
4382 \@@_draw_line:
4383 }
4384 \cs_new_protected:Npn \@@_open_y_initial_dim:
4385 {
4386 \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4387 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4388 {
4389 \cs_if_exist:cT
4390 { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4391 {
4392 \pgfpointanchor
4393 { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4394 { north }
4395 \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4396 { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4397 }
4398 }
4399 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4400 {
4401 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4402 \dim_set:Nn \l_@@_y_initial_dim
4403 {
4404 \fp_to_dim:n
4405 {
4406 \pgf@y
4407 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4408 }
4409 }
4410 }
4411 }
4412 \cs_new_protected:Npn \@@_open_y_final_dim:
4413 {
4414 \dim_set_eq:NN \l_@@_y_final_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_@@_final_i_int - ##1 }
4419 {
4420 \pgfpointanchor
4421 { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4422 { south }
4423 \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4424 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4425 }
4426 }

```

```

4427 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4428 {
4429   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4430   \dim_set:Nn \l_@@_y_final_dim
4431     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4432 }
4433 }

```

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.

```

4434 \cs_new_protected:Npn \@@_draw_Vdots:nmm #1 #2 #3
4435 {
4436   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4437   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4438   {
4439     \@@_find_extremities_of_line:nmmm { #1 } { #2 } 1 0

```

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

```

4440   \group_begin:
4441     \@@_open_shorten:
4442     \int_if_zero:nTF { #2 }
4443       { \color { nicematrix-first-col } }
4444       {
4445         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4446           { \color { nicematrix-last-col } }
4447       }
4448     \keys_set:nn { nicematrix / xdots } { #3 }
4449     \@@_color:o \l_@@_xdots_color_tl
4450     \@@_actually_draw_Vdots:
4451   \group_end:
4452 }
4453 }

```

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`.

```

4454 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4455 {

```

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

```

4456   \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.

```

4457   {
4458     \@@_open_y_initial_dim:
4459     \@@_open_y_final_dim:
4460     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4461     {
4462       \@@_qpoint:n { col - 1 }
4463       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4464       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim

```

```

4465     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4466     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4467   }
4468   {
4469     \bool_lazy_and:nnTF
4470     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4471     { \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”.

```

4472     {
4473       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4474       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4475       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4476       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4477       \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4478     }

```

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

```

4479     {
4480       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4481       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4482       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4483       \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4484     }
4485   }
4486 }

```

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

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

```

4487   {
4488     \bool_set_false:N \l_tmpa_bool
4489     \bool_if:NF \l_@@_initial_open_bool
4490     {
4491       \bool_if:NF \l_@@_final_open_bool
4492       {
4493         \@@_set_initial_coords_from_anchor:n { south-west }
4494         \@@_set_final_coords_from_anchor:n { north-west }
4495         \bool_set:Nn \l_tmpa_bool
4496         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4497       }
4498     }

```

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

```

4499     \bool_if:NTF \l_@@_initial_open_bool
4500     {
4501       \@@_open_y_initial_dim:
4502       \@@_set_final_coords_from_anchor:n { north }
4503       \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4504     }
4505     {
4506       \@@_set_initial_coords_from_anchor:n { south }
4507       \bool_if:NTF \l_@@_final_open_bool
4508       \@@_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.

```

4509     {
4510       \@@_set_final_coords_from_anchor:n { north }
4511       \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4512       {
4513         \dim_set:Nn \l_@@_x_initial_dim
4514         {
4515           \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4516           \l_@@_x_initial_dim \l_@@_x_final_dim
4517         }

```

```

4518         }
4519     }
4520 }
4521 }
4522 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4523 \@@_draw_line:
4524 }

```

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.

```

4525 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4526 {
4527     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4528     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4529     {
4530         \@@_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.

```

4531         \group_begin:
4532         \@@_open_shorten:
4533         \keys_set:nn { nicematrix / xdots } { #3 }
4534         \@@_color:o \l_@@_xdots_color_tl
4535         \@@_actually_draw_Ddots:
4536     \group_end:
4537 }
4538 }

```

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`.

```

4539 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4540 {
4541     \bool_if:NTF \l_@@_initial_open_bool
4542     {
4543         \@@_open_y_initial_dim:
4544         \@@_open_x_initial_dim:
4545     }
4546     { \@@_set_initial_coords_from_anchor:n { south-east } }
4547     \bool_if:NTF \l_@@_final_open_bool
4548     {
4549         \@@_open_x_final_dim:
4550         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4551     }
4552     { \@@_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.

```

4553     \bool_if:NT \l_@@_parallelize_diags_bool
4554     {
4555         \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).

```
4556 \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.

```
4557 {
4558   \dim_gset:Nn \g_@@_delta_x_one_dim
4559   { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4560   \dim_gset:Nn \g_@@_delta_y_one_dim
4561   { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4562 }
```

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`.

```
4563 {
4564   \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4565   {
4566     \dim_set:Nn \l_@@_y_final_dim
4567     {
4568       \l_@@_y_initial_dim +
4569       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4570       \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4571     }
4572   }
4573 }
4574 }
4575 \@@_draw_line:
4576 }
```

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.

```
4577 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4578 {
4579   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4580   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4581   {
4582     \@@_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.

```
4583   \group_begin:
4584   \@@_open_shorten:
4585   \keys_set:nn { nicematrix / xdots } { #3 }
4586   \@@_color:o \l_@@_xdots_color_tl
4587   \@@_actually_draw_Iddots:
4588   \group_end:
4589 }
4590 }
```

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`.

```

4591 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4592 {
4593   \bool_if:NTF \l_@@_initial_open_bool
4594   {
4595     \@@_open_y_initial_dim:
4596     \@@_open_x_initial_dim:
4597   }
4598   { \@@_set_initial_coords_from_anchor:n { south-west } }
4599   \bool_if:NTF \l_@@_final_open_bool
4600   {
4601     \@@_open_y_final_dim:
4602     \@@_open_x_final_dim:
4603   }
4604   { \@@_set_final_coords_from_anchor:n { north-east } }
4605   \bool_if:NT \l_@@_parallelize_diags_bool
4606   {
4607     \int_gincr:N \g_@@_iddots_int
4608     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4609     {
4610       \dim_gset:Nn \g_@@_delta_x_two_dim
4611       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4612       \dim_gset:Nn \g_@@_delta_y_two_dim
4613       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4614     }
4615     {
4616       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4617       {
4618         \dim_set:Nn \l_@@_y_final_dim
4619         {
4620           \l_@@_y_initial_dim +
4621           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4622           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4623         }
4624       }
4625     }
4626   }
4627   \@@_draw_line:
4628 }

```

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`

```

4629 \cs_new_protected:Npn \@@_draw_line:
4630 {
4631   \pgfrememberpicturepositiononpagetrue
4632   \pgf@relevantforpicturesizefalse

```

```

4633 \bool_lazy_or:nnTF
4634   { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4635   \l_@@_dotted_bool
4636   \@@_draw_standard_dotted_line:
4637   \@@_draw_unstandard_dotted_line:
4638 }

```

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.

```

4639 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4640 {
4641   \begin { scope }
4642   \@@_draw_unstandard_dotted_line:o
4643   { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4644 }

```

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.

```

4645 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4646 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4647 {
4648   \@@_draw_unstandard_dotted_line:nooo
4649   { #1 }
4650   \l_@@_xdots_up_tl
4651   \l_@@_xdots_down_tl
4652   \l_@@_xdots_middle_tl
4653 }

```

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

```

4654 \hook_gput_code:nnn { begindocument } { . }
4655 {
4656   \IfPackageLoadedT { tikz }
4657   {
4658     \tikzset
4659     {
4660       @@_node_above / .style = { sloped , above } ,
4661       @@_node_below / .style = { sloped , below } ,
4662       @@_node_middle / .style =
4663       {
4664         sloped ,
4665         inner~sep = \c_@@_innersep_middle_dim
4666       }
4667     }
4668   }
4669 }

```

```

4670 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { n o o o }
4671 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n { n o o o } #1 #2 #3 #4
4672 {

```

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.

```

4673 \dim_zero_new:N \l_@@_l_dim
4674 \dim_set:Nn \l_@@_l_dim
4675 {
4676   \fp_to_dim:n

```

```

4677     {
4678         sqrt
4679         (
4680             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4681             +
4682             ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4683         )
4684     }
4685 }

```

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.

```

4686     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4687     {
4688         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4689         \@@_draw_unstandard_dotted_line_i:
4690     }

```

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

```

4691     \bool_if:NT \l_@@_xdots_h_labels_bool
4692     {
4693         \tikzset
4694         {
4695             @@_node_above / .style = { auto = left } ,
4696             @@_node_below / .style = { auto = right } ,
4697             @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4698         }
4699     }
4700     \tl_if_empty:nF { #4 }
4701     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4702     \draw
4703     [ #1 ]
4704     ( \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).

```

4705     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4706     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4707     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4708     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4709     \end { scope }
4710 }

4711 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4712 {
4713     \dim_set:Nn \l_tmpa_dim
4714     {
4715         \l_@@_x_initial_dim
4716         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4717         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4718     }
4719     \dim_set:Nn \l_tmpb_dim
4720     {
4721         \l_@@_y_initial_dim
4722         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4723         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4724     }
4725     \dim_set:Nn \l_@@_tmpc_dim
4726     {
4727         \l_@@_x_final_dim
4728         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4729         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4730     }

```

```

4731 \dim_set:Nn \l_@@_tmpd_dim
4732 {
4733   \l_@@_y_final_dim
4734   - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4735   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4736 }
4737 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4738 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4739 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4740 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4741 }

```

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

```

4742 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4743 {
4744   \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.

```

4745   \dim_zero_new:N \l_@@_l_dim
4746   \dim_set:Nn \l_@@_l_dim
4747   {
4748     \fp_to_dim:n
4749     {
4750       sqrt
4751       (
4752         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4753         +
4754         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4755       )
4756     }
4757   }

```

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.

```

4758   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4759   {
4760     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4761     \@@_draw_standard_dotted_line_i:
4762   }
4763   \group_end:
4764   \bool_lazy_all:nF
4765   {
4766     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4767     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4768     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4769   }
4770   \l_@@_labels_standard_dotted_line:
4771 }
4772 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4773 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4774 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4775   \int_set:Nn \l_tmpa_int
4776   {
4777     \dim_ratio:nn
4778     {
4779       \l_@@_l_dim

```

```

4780         - \l_@@_xdots_shorten_start_dim
4781         - \l_@@_xdots_shorten_end_dim
4782     }
4783     \l_@@_xdots_inter_dim
4784 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4785 \dim_set:Nn \l_tmpa_dim
4786 {
4787   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4788   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4789 }
4790 \dim_set:Nn \l_tmpb_dim
4791 {
4792   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4793   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4794 }

```

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.

```

4795 \dim_gadd:Nn \l_@@_x_initial_dim
4796 {
4797   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4798   \dim_ratio:nn
4799   {
4800     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4801     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4802   }
4803   { 2 \l_@@_l_dim }
4804 }
4805 \dim_gadd:Nn \l_@@_y_initial_dim
4806 {
4807   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4808   \dim_ratio:nn
4809   {
4810     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4811     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4812   }
4813   { 2 \l_@@_l_dim }
4814 }
4815 \pgf@relevantforpicturesizefalse
4816 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4817 {
4818   \pgfpathcircle
4819   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4820   { \l_@@_xdots_radius_dim }
4821   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4822   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4823 }
4824 \pgfusepathqfill
4825 }

```

```

4826 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4827 {
4828   \pgfscope
4829   \pgftransformshift
4830   {
4831     \pgfpointlineattime { 0.5 }
4832     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4833     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4834   }
4835   \fp_set:Nn \l_tmpa_fp

```

```

4836 {
4837   atand
4838   (
4839     \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4840     \l_@@_x_final_dim - \l_@@_x_initial_dim
4841   )
4842 }
4843 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4844 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4845 \tl_if_empty:NF \l_@@_xdots_middle_tl
4846 {
4847   \begin { pgfscope }
4848   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4849   \pgfnode
4850   { rectangle }
4851   { center }
4852   {
4853     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4854     {
4855       \c_math_toggle_token
4856       \scriptstyle \l_@@_xdots_middle_tl
4857       \c_math_toggle_token
4858     }
4859   }
4860   { }
4861   {
4862     \pgfsetfillcolor { white }
4863     \pgfusepath { fill }
4864   }
4865   \end { pgfscope }
4866 }
4867 \tl_if_empty:NF \l_@@_xdots_up_tl
4868 {
4869   \pgfnode
4870   { rectangle }
4871   { south }
4872   {
4873     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4874     {
4875       \c_math_toggle_token
4876       \scriptstyle \l_@@_xdots_up_tl
4877       \c_math_toggle_token
4878     }
4879   }
4880   { }
4881   { \pgfusepath { } }
4882 }
4883 \tl_if_empty:NF \l_@@_xdots_down_tl
4884 {
4885   \pgfnode
4886   { rectangle }
4887   { north }
4888   {
4889     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4890     {
4891       \c_math_toggle_token
4892       \scriptstyle \l_@@_xdots_down_tl
4893       \c_math_toggle_token
4894     }
4895   }
4896   { }
4897   { \pgfusepath { } }
4898 }

```

```

4899   \endpgfscope
4900 }

```

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.

```

4901 \hook_gput_code:nnn { begindocument } { . }
4902 {
4903   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4904   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4905   \cs_new_protected:Npn \@@_Ldots
4906     { \@@_collect_options:n { \@@_Ldots_i } }
4907   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4908     {
4909       \int_if_zero:nTF \c@jCol
4910         { \@@_error:nn { in~first~col } \Ldots }
4911         {
4912           \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4913             { \@@_error:nn { in~last~col } \Ldots }
4914             {
4915               \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4916               { #1 , down = #2 , up = #3 , middle = #4 }
4917             }
4918         }
4919       \bool_if:NF \l_@@_nullify_dots_bool
4920         { \phantom { \ensuremath { \@@_old_ldots } } }
4921       \bool_gset_true:N \g_@@_empty_cell_bool
4922     }

4923   \cs_new_protected:Npn \@@_Cdots
4924     { \@@_collect_options:n { \@@_Cdots_i } }
4925   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4926     {
4927       \int_if_zero:nTF \c@jCol
4928         { \@@_error:nn { in~first~col } \Cdots }
4929         {
4930           \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4931             { \@@_error:nn { in~last~col } \Cdots }
4932             {
4933               \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4934               { #1 , down = #2 , up = #3 , middle = #4 }
4935             }
4936         }
4937       \bool_if:NF \l_@@_nullify_dots_bool
4938         { \phantom { \ensuremath { \@@_old_cdots } } }
4939       \bool_gset_true:N \g_@@_empty_cell_bool
4940     }

```

```

4941 \cs_new_protected:Npn \@@_Vdots
4942   { \@@_collect_options:n { \@@_Vdots_i } }
4943 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4944   {
4945     \int_if_zero:nTF \c@iRow
4946       { \@@_error:nn { in~first~row } \Vdots }
4947     {
4948       \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4949         { \@@_error:nn { in~last~row } \Vdots }
4950       {
4951         \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4952         { #1 , down = #2 , up = #3 , middle = #4 }
4953       }
4954     }
4955     \bool_if:NF \l_@@_nullify_dots_bool
4956     { \phantom { \ensuremath { \@@_old_vdots } } }
4957     \bool_gset_true:N \g_@@_empty_cell_bool
4958   }

4959 \cs_new_protected:Npn \@@_Ddots
4960   { \@@_collect_options:n { \@@_Ddots_i } }
4961 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4962   {
4963     \int_case:nnF \c@iRow
4964     {
4965       0 { \@@_error:nn { in~first~row } \Ddots }
4966       \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4967     }
4968     {
4969       \int_case:nnF \c@jCol
4970       {
4971         0 { \@@_error:nn { in~first~col } \Ddots }
4972         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4973       }
4974       {
4975         \keys_set_known:nn { nicematrix / Ddots } { #1 }
4976         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4977         { #1 , down = #2 , up = #3 , middle = #4 }
4978       }
4979     }
4980     \bool_if:NF \l_@@_nullify_dots_bool
4981     { \phantom { \ensuremath { \@@_old_ddots } } }
4982     \bool_gset_true:N \g_@@_empty_cell_bool
4983   }
4984 }

4985 \cs_new_protected:Npn \@@_Iddots
4986   { \@@_collect_options:n { \@@_Iddots_i } }
4987 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4988   {
4989     \int_case:nnF \c@iRow
4990     {
4991       0 { \@@_error:nn { in~first~row } \Iddots }
4992       \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4993     }
4994     {
4995       \int_case:nnF \c@jCol
4996       {
4997         0 { \@@_error:nn { in~first~col } \Iddots }
4998         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4999       }
5000     }

```

```

5001         \keys_set_known:nm { nicematrix / Ddots } { #1 }
5002         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5003             { #1 , down = #2 , up = #3 , middle = #4 }
5004     }
5005 }
5006 \bool_if:NF \l_@@_nullify_dots_bool
5007 { \phantom { \ensuremath { \@@_old_iddots } } }
5008 \bool_gset_true:N \g_@@_empty_cell_bool
5009 }
5010 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5011 \keys_define:nm { nicematrix / Ddots }
5012 {
5013     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5014     draw-first .default:n = true ,
5015     draw-first .value_forbidden:n = true
5016 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5017 \cs_new_protected:Npn \@@_Hspace:
5018 {
5019     \bool_gset_true:N \g_@@_empty_cell_bool
5020     \hspace
5021 }

```

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`.

```

5022 \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`.

```

5023 \cs_new:Npn \@@_Hdotsfor:
5024 {
5025     \bool_lazy_and:nnTF
5026         { \int_if_zero_p:n \c@jCol }
5027         { \int_if_zero_p:n \l_@@_first_col_int }
5028     {
5029         \bool_if:NTF \g_@@_after_col_zero_bool
5030         {
5031             \multicolumn { 1 } { c } { }
5032             \@@_Hdotsfor_i
5033         }
5034         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5035     }
5036     {
5037         \multicolumn { 1 } { c } { }
5038         \@@_Hdotsfor_i
5039     }
5040 }

```

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:`).

```

5041 \hook_gput_code:nnn { begindocument } { . }
5042 {
5043     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5044     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put ! before the last optional argument for homogeneity with \Cdots, etc. which have only one optional argument.

```

5045 \cs_new_protected:Npn \@@_Hdotsfor_i
5046 { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5047 \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5048 {
5049   \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5050   {
5051     \@@_Hdotsfor:nnnn
5052     { \int_use:N \c@iRow }
5053     { \int_use:N \c@jCol }
5054     { #2 }
5055     {
5056       #1 , #3 ,
5057       down = \exp_not:n { #4 } ,
5058       up = \exp_not:n { #5 } ,
5059       middle = \exp_not:n { #6 }
5060     }
5061   }
5062   \prg_replicate:nn { #2 - 1 }
5063   {
5064     &
5065     \multicolumn { 1 } { c } { }
5066     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5067   }
5068 }
5069 }

```

```

5070 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5071 {
5072   \bool_set_false:N \l_@@_initial_open_bool
5073   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5074 \int_set:Nn \l_@@_initial_i_int { #1 }
5075 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5076 \int_compare:nNnTF { #2 } = \c_one_int
5077 {
5078   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5079   \bool_set_true:N \l_@@_initial_open_bool
5080 }
5081 {
5082   \cs_if_exist:cTF
5083   {
5084     pgf @ sh @ ns @ \@@_env:
5085     - \int_use:N \l_@@_initial_i_int
5086     - \int_eval:n { #2 - 1 }
5087   }
5088   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5089   {
5090     \int_set:Nn \l_@@_initial_j_int { #2 }
5091     \bool_set_true:N \l_@@_initial_open_bool
5092   }
5093 }
5094 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5095 {
5096   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5097   \bool_set_true:N \l_@@_final_open_bool
5098 }
5099 {
5100   \cs_if_exist:cTF
5101   {

```

```

5102     pgf @ sh @ ns @ \@@_env:
5103     - \int_use:N \l_@@_final_i_int
5104     - \int_eval:n { #2 + #3 }
5105     }
5106     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5107     {
5108         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5109         \bool_set_true:N \l_@@_final_open_bool
5110     }
5111 }
5112 \group_begin:
5113 \@@_open_shorten:
5114 \int_if_zero:nTF { #1 }
5115   { \color { nicematrix-first-row } }
5116   {
5117       \int_compare:nNnT { #1 } = \g_@@_row_total_int
5118       { \color { nicematrix-last-row } }
5119   }
5120
5121 \keys_set:nn { nicematrix / xdots } { #4 }
5122 \@@_color:o \l_@@_xdots_color_tl
5123 \@@_actually_draw_Ldots:
5124 \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).

```

5125     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5126     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5127 }

5128 \hook_gput_code:nnn { begindocument } { . }
5129 {
5130     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5131     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5132     \cs_new_protected:Npn \@@_Vdotsfor:
5133       { \@@_collect_options:n { \@@_Vdotsfor_i } }
5134     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5135       {
5136         \bool_gset_true:N \g_@@_empty_cell_bool
5137         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5138         {
5139             \@@_Vdotsfor:nnnn
5140             { \int_use:N \c@iRow }
5141             { \int_use:N \c@jCol }
5142             { #2 }
5143             {
5144                 #1 , #3 ,
5145                 down = \exp_not:n { #4 } ,
5146                 up = \exp_not:n { #5 } ,
5147                 middle = \exp_not:n { #6 }
5148             }
5149         }
5150     }
5151 }

5152 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5153 {
5154     \bool_set_false:N \l_@@_initial_open_bool
5155     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```
5156 \int_set:Nn \l_@@_initial_j_int { #2 }
5157 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

For the row, it's a bit more complicated.

```
5158 \int_compare:nNnTF { #1 } = \c_one_int
5159 {
5160   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5161   \bool_set_true:N \l_@@_initial_open_bool
5162 }
5163 {
5164   \cs_if_exist:cTF
5165   {
5166     pgf @ sh @ ns @ \@@_env:
5167     - \int_eval:n { #1 - 1 }
5168     - \int_use:N \l_@@_initial_j_int
5169   }
5170   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5171   {
5172     \int_set:Nn \l_@@_initial_i_int { #1 }
5173     \bool_set_true:N \l_@@_initial_open_bool
5174   }
5175 }
5176 \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5177 {
5178   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5179   \bool_set_true:N \l_@@_final_open_bool
5180 }
5181 {
5182   \cs_if_exist:cTF
5183   {
5184     pgf @ sh @ ns @ \@@_env:
5185     - \int_eval:n { #1 + #3 }
5186     - \int_use:N \l_@@_final_j_int
5187   }
5188   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5189   {
5190     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5191     \bool_set_true:N \l_@@_final_open_bool
5192   }
5193 }
5194 \group_begin:
5195 \@@_open_shorten:
5196 \int_if_zero:nTF { #2 }
5197 { \color { nicematrix-first-col } }
5198 {
5199   \int_compare:nNnT { #2 } = \g_@@_col_total_int
5200   { \color { nicematrix-last-col } }
5201 }
5202 \keys_set:mn { nicematrix / xdots } { #4 }
5203 \@@_color:o \l_@@_xdots_color_tl
5204 \@@_actually_draw_Vdots:
5205 \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).

```
5206 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5207 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5208 }
```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5209 \NewDocumentCommand \@@_rotate: { 0 { } }
5210 {
5211   \peek_remove_spaces:n
5212   {
5213     \bool_gset_true:N \g_@@_rotate_bool
5214     \keys_set:nn { nicematrix / rotate } { #1 }
5215   }
5216 }

5217 \keys_define:nn { nicematrix / rotate }
5218 {
5219   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5220   c .value_forbidden:n = true ,
5221   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5222 }

```

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

```

5223 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5224 {
5225   \tl_if_empty:nTF { #2 }
5226   { #1 }
5227   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5228 }
5229 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5230 { \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).

```

5231 \hook_gput_code:nnn { begindocument } { . }
5232 {
5233   \cs_set_nopar:Npn \l_@@_argspec_tl
5234   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5235   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5236   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5237   {
5238     \group_begin:
5239     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5240     \@@_color:o \l_@@_xdots_color_tl
5241     \use:e

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5242     {
5243         \@@_line_i:nn
5244         { \@@_double_int_eval:n #2 - \q_stop }
5245         { \@@_double_int_eval:n #3 - \q_stop }
5246     }
5247     \group_end:
5248 }
5249 }

5250 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5251 {
5252     \bool_set_false:N \l_@@_initial_open_bool
5253     \bool_set_false:N \l_@@_final_open_bool
5254     \bool_lazy_or:nnTF
5255     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5256     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5257     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5258     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5259 }

5260 \hook_gput_code:nnn { begindocument } { . }
5261 {
5262     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5263     {

```

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:`.

```

5264     \c_@@_pgfortikzpicture_tl
5265     \@@_draw_line_iii:nn { #1 } { #2 }
5266     \c_@@_endpgfortikzpicture_tl
5267 }
5268 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5269 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5270 {
5271     \pgfrememberpicturepositiononpagetrue
5272     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5273     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5274     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5275     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5276     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5277     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5278     \@@_draw_line:
5279 }

```

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`

```
5280 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5281 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5282 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5283 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5284 {
5285   \tl_gput_right:Ne \g_@@_row_style_tl
5286   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5287   \exp_not:N
5288   \@@_if_row_less_than:nn
5289   { \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`).

```
5290   { \exp_not:n { #1 } \scan_stop: }
5291 }
5292 }
```

```
5293 \keys_define:nn { nicematrix / RowStyle }
5294 {
5295   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5296   cell-space-top-limit .value_required:n = true ,
5297   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5298   cell-space-bottom-limit .value_required:n = true ,
5299   cell-space-limits .meta:n =
5300   {
5301     cell-space-top-limit = #1 ,
5302     cell-space-bottom-limit = #1 ,
5303   } ,
5304   color .tl_set:N = \l_@@_color_tl ,
5305   color .value_required:n = true ,
5306   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5307   bold .default:n = true ,
5308   nb-rows .code:n =
5309   \str_if_eq:eeTF { #1 } { * }
5310   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5311   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5312   nb-rows .value_required:n = true ,
5313   fill .tl_set:N = \l_@@_fill_tl ,
5314   fill .value_required:n = true ,
5315   opacity .tl_set:N = \l_@@_opacity_tl ,
5316   opacity .value_required:n = true ,
5317   rowcolor .tl_set:N = \l_@@_fill_tl ,
5318   rowcolor .value_required:n = true ,
5319   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5320   rounded-corners .default:n = 4 pt ,
5321   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5322 }
```

```
5323 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5324 {
5325   \group_begin:
```

```

5326 \tl_clear:N \l_@@_fill_tl
5327 \tl_clear:N \l_@@_opacity_tl
5328 \tl_clear:N \l_@@_color_tl
5329 \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5330 \dim_zero:N \l_@@_rounded_corners_dim
5331 \dim_zero:N \l_tmpa_dim
5332 \dim_zero:N \l_tmpb_dim
5333 \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key rowcolor (of its alias fill) has been used.

```

5334 \tl_if_empty:NF \l_@@_fill_tl
5335 {
5336   \@@_add_opacity_to_fill:
5337   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5338   {

```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```

5339 \int_compare:nNnTF \c@jCol > \c_one_int
5340 {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row). The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5341 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5342 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5343 { \int_use:N \c@iRow - * }
5344 { \dim_use:N \l_@@_rounded_corners_dim }

```

Then, the other rows (if there are several rows).

```

5345 \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5346 { \@@_rounded_from_row:n { \c@iRow + 1 } }
5347 }

```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```

5348 { \@@_rounded_from_row:n { \c@iRow } }
5349 }
5350 }
5351 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5352 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5353 {
5354   \@@_put_in_row_style:e
5355   {
5356     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5357     {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5358 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5359 { \dim_use:N \l_tmpa_dim }
5360 }
5361 }
5362 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5363 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5364 {
5365   \@@_put_in_row_style:e
5366   {
5367     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5368     {
5369       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5370       { \dim_use:N \l_tmpb_dim }
5371     }
5372   }
5373 }

```

`\l_@@_color_tl` is the value of the key color of `\RowStyle`.

```

5374   \tl_if_empty:NF \l_@@_color_tl
5375   {
5376     \@@_put_in_row_style:e
5377     {
5378       \mode_leave_vertical:
5379       \@@_color:n { \l_@@_color_tl }
5380     }
5381   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5382   \bool_if:NT \l_@@_bold_row_style_bool
5383   {
5384     \@@_put_in_row_style:n
5385     {
5386       \exp_not:n
5387       {
5388         \if_mode_math:
5389         \c_math_toggle_token
5390         \bfseries \boldmath
5391         \c_math_toggle_token
5392       \else:
5393         \bfseries \boldmath
5394       \fi:
5395     }
5396   }
5397 }
5398 \group_end:
5399 \g_@@_row_style_tl
5400 \ignorespaces
5401 }

```

The following command must *not* be protected.

```

5402 \cs_new:Npn \@@_rounded_from_row:n #1
5403 {
5404   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5405   { \int_eval:n { #1 } - 1 }
5406   {
5407     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5408     - \exp_not:n { \int_use:N \c@jCol }
5409   }
5410   { \dim_use:N \l_@@_rounded_corners_dim }
5411 }

```

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).

```
5412 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5413 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5414 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5415 {
```

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.

```
5416 \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`.

```
5417 \str_if_in:nnF { #1 } { !! }
5418 {
5419 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5420 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5421 }
5422 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5423 {
5424 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5425 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5426 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5427 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5428 }
```

The following command must be used within a `\pgfpicture`.

```
5429 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5430 {
5431 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5432 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5433 \group_begin:
5434 \pgfsetcornersarced
5435 {
5436 \pgfpoint
5437 { \l_@@_tab_rounded_corners_dim }
5438 { \l_@@_tab_rounded_corners_dim }
5439 }
```

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.

```
5440 \bool_if:NTF \l_@@_hvlines_bool
5441 {
5442 \pgfpathrectanglecorners
```

```

5443     {
5444         \pgfpointadd
5445         { \@@_qpoint:n { row-1 } }
5446         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5447     }
5448     {
5449         \pgfpointadd
5450         {
5451             \@@_qpoint:n
5452             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5453         }
5454         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5455     }
5456 }
5457 {
5458     \pgfpathrectanglecorners
5459     { \@@_qpoint:n { row-1 } }
5460     {
5461         \pgfpointadd
5462         {
5463             \@@_qpoint:n
5464             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5465         }
5466         { \pgfpoint \c_zero_dim \arrayrulewidth }
5467     }
5468 }
5469 \pgfusepath { clip }
5470 \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5471     }
5472 }

```

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`).

```

5473 \cs_new_protected:Npn \@@_actually_color:
5474 {
5475     \pgfpicture
5476     \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.

```

5477     \@@_clip_with_rounded_corners:
5478     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5479     {
5480         \int_compare:nNnTF { ##1 } = \c_one_int
5481         {
5482             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5483             \use:c { g_@@_color _ 1 _tl }
5484             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5485         }
5486         {
5487             \begin { pgfscope }
5488                 \@@_color_opacity ##2
5489                 \use:c { g_@@_color _ ##1 _tl }
5490                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5491                 \pgfusepath { fill }
5492             \end { pgfscope }
5493         }
5494     }
5495 \endpgfpicture
5496 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5497 \cs_new_protected:Npn \@@_color_opacity
5498 {
5499   \peek_meaning:NTF [
5500     { \@@_color_opacity:w }
5501     { \@@_color_opacity:w [ ] }
5502   }

```

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.

```

5503 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5504 {
5505   \tl_clear:N \l_tmpa_tl
5506   \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.

```

5507   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5508   \tl_if_empty:NTF \l_tmpb_tl
5509     { \@declaredcolor }
5510     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5511 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5512 \keys_define:nn { nicematrix / color-opacity }
5513 {
5514   opacity .tl_set:N          = \l_tmpa_tl ,
5515   opacity .value_required:n = true
5516 }

5517 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5518 {
5519   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5520   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5521   \@@_cartesian_path:
5522 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5523 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5524 {
5525   \tl_if_blank:nF { #2 }
5526   {
5527     \@@_add_to_colors_seq:en
5528     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5529     { \@@_cartesian_color:nn { #3 } { - } }
5530   }
5531 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5532 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5533 {
5534   \tl_if_blank:nF { #2 }
5535   {
5536     \@@_add_to_colors_seq:en
5537     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5538     { \@@_cartesian_color:nn { - } { #3 } }
5539   }
5540 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5541 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5542 {
5543   \tl_if_blank:nF { #2 }
5544   {
5545     \@@_add_to_colors_seq:en
5546     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5547     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5548   }
5549 }

```

The last argument is the radius of the corners of the rectangle.

```

5550 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5551 {
5552   \tl_if_blank:nF { #2 }
5553   {
5554     \@@_add_to_colors_seq:en
5555     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5556     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5557   }
5558 }

```

The last argument is the radius of the corners of the rectangle.

```

5559 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5560 {
5561   \@@_cut_on_hyphen:w #1 \q_stop
5562   \tl_clear_new:N \l_@@_tmpc_tl
5563   \tl_clear_new:N \l_@@_tmpd_tl
5564   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5565   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5566   \@@_cut_on_hyphen:w #2 \q_stop
5567   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5568   \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`.

```

5569   \@@_cartesian_path:n { #3 }
5570 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5571 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5572 {
5573   \clist_map_inline:nn { #3 }
5574   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5575 }

```

```

5576 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5577 {
5578   \int_step_inline:nn \c@iRow
5579   {
5580     \int_step_inline:nn \c@jCol
5581     {
5582       \int_if_even:nTF { #####1 + ##1 }
5583       { \@@_cellcolor [ #1 ] { #2 } }
5584       { \@@_cellcolor [ #1 ] { #3 } }
5585       { ##1 - #####1 }
5586     }
5587   }
5588 }

```

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”.

```

5589 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5590 {
5591   \@@_rectanglecolor [ #1 ] { #2 }
5592   { 1 - 1 }
5593   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5594 }

5595 \keys_define:nn { nicematrix / rowcolors }
5596 {
5597   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5598   respect-blocks .default:n = true ,
5599   cols .tl_set:N = \l_@@_cols_tl ,
5600   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5601   restart .default:n = true ,
5602   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5603 }

```

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*.

```

5604 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5605 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5606   \group_begin:
5607   \seq_clear_new:N \l_@@_colors_seq
5608   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5609   \tl_clear_new:N \l_@@_cols_tl
5610   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5611   \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).

```

5612   \int_zero_new:N \l_@@_color_int
5613   \int_set_eq:NN \l_@@_color_int \c_one_int
5614   \bool_if:NT \l_@@_respect_blocks_bool
5615   {

```

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`).

```

5616     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5617     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5618     { \@@_not_in_exterior_p:nnnn ##1 }
5619   }
5620   \pgfpicture
5621   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5622   \clist_map_inline:nn { #2 }
5623   {
5624     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5625     \tl_if_in:NnTF \l_tmpa_tl { - }
5626     { \@@_cut_on_hyphen:w ##1 \q_stop }
5627     { \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.

```

5628     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5629     \int_set:Nn \l_@@_color_int
5630     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5631     \int_zero_new:N \l_@@_tmpc_int
5632     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5633     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5634     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5635         \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).

```

5636         \bool_if:NT \l_@@_respect_blocks_bool
5637         {
5638             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5639             { \@@_intersect_our_row_p:nnnnn ###1 }
5640             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5641         }
5642         \tl_set:No \l_@@_rows_tl
5643         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5644         \tl_clear_new:N \l_@@_color_tl
5645         \tl_set:Ne \l_@@_color_tl
5646         {
5647             \@@_color_index:n
5648             {
5649                 \int_mod:nn
5650                 { \l_@@_color_int - 1 }
5651                 { \seq_count:N \l_@@_colors_seq }
5652                 + 1
5653             }
5654         }
5655         \tl_if_empty:NF \l_@@_color_tl
5656         {
5657             \@@_add_to_colors_seq:ee
5658             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5659             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5660         }
5661         \int_incr:N \l_@@_color_int
5662         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5663     }
5664 }
5665 \endpgfpicture
5666 \group_end:
5667 }

```

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.

```

5668 \cs_new:Npn \@@_color_index:n #1
5669 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5670     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5671     { \@@_color_index:n { #1 - 1 } }
5672     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5673 }

```

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.

```
5674 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5675 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5676 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5677 {
5678   \int_compare:nNnT { #3 } > \l_tmpb_int
5679   { \int_set:Nn \l_tmpb_int { #3 } }
5680 }
```

```
5681 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5682 {
5683   \int_if_zero:nTF { #4 }
5684   \prg_return_false:
5685   {
5686     \int_compare:nNnTF { #2 } > \c@jCol
5687     \prg_return_false:
5688     \prg_return_true:
5689   }
5690 }
```

The following command return true when the block intersects the row `\l_tmpa_int`.

```
5691 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5692 {
5693   \int_compare:nNnTF { #1 } > \l_tmpa_int
5694   \prg_return_false:
5695   {
5696     \int_compare:nNnTF \l_tmpa_int > { #3 }
5697     \prg_return_false:
5698     \prg_return_true:
5699   }
5700 }
```

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`).

```
5701 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5702 {
5703   \dim_compare:nNnTF { #1 } = \c_zero_dim
5704   {
5705     \bool_if:NTF
5706     \l_@@_nocolor_used_bool
5707     \@@_cartesian_path_normal_ii:
5708     {
5709       \clist_if_empty:NTF \l_@@_corners_cells_clist
5710       { \@@_cartesian_path_normal_i:n { #1 } }
5711       \@@_cartesian_path_normal_ii:
5712     }
5713   }
5714   { \@@_cartesian_path_normal_i:n { #1 } }
5715 }
```

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.

```
5716 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5717 {
5718   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```

We begin the loop over the columns.

```
5719 \clist_map_inline:Nn \l_@@_cols_tl
5720 {
5721   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5722   \tl_if_in:NnTF \l_tmpa_tl { - }
5723     { \@@_cut_on_hyphen:w ##1 \q_stop }
5724     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5725   \tl_if_empty:NTF \l_tmpa_tl
5726     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5727     {
5728       \str_if_eq:eeT \l_tmpa_tl { * }
5729       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5730     }
5731   \tl_if_empty:NTF \l_tmpb_tl
5732     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5733     {
5734       \str_if_eq:eeT \l_tmpb_tl { * }
5735       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5736     }
5737   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5738     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```

`\l_@@_tmpc_tl` will contain the number of column.

```
5739   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5740   \@@_qpoint:n { col - \l_tmpa_tl }
5741   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5742     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5743     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5744   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5745   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5746 \clist_map_inline:Nn \l_@@_rows_tl
5747 {
5748   \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5749   \tl_if_in:NnTF \l_tmpa_tl { - }
5750     { \@@_cut_on_hyphen:w ####1 \q_stop }
5751     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5752   \tl_if_empty:NTF \l_tmpa_tl
5753     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5754     {
5755       \str_if_eq:eeT \l_tmpa_tl { * }
5756       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5757     }
5758   \tl_if_empty:NTF \l_tmpb_tl
5759     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5760     {
5761       \str_if_eq:eeT \l_tmpb_tl { * }
5762       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5763     }
5764   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5765     { \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`.

```
5766   \cs_if_exist:cF
5767     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5768     {
5769       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5770       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
```

```

5771         \@@_qpoint:n { row - \l_tmpa_tl }
5772         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5773         \pgfpathrectanglecorners
5774         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5775         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5776     }
5777 }
5778 }
5779 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5780 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5781 {
5782     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5783     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5784     \clist_map_inline:Nn \l_@@_cols_tl
5785     {
5786         \@@_qpoint:n { col - ##1 }
5787         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5788         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5789         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5790         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5791         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5792     \clist_map_inline:Nn \l_@@_rows_tl
5793     {
5794         \@@_if_in_corner:nF { ##### - ##1 }
5795         {
5796             \@@_qpoint:n { row - \int_eval:n { ##### + 1 } }
5797             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5798             \@@_qpoint:n { row - ##### }
5799             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5800             \cs_if_exist:cF { @@_nocolor_##### - ##1 }
5801             {
5802                 \pgfpathrectanglecorners
5803                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5804                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5805             }
5806         }
5807     }
5808 }
5809 }

```

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`).

```

5810 \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

```

5811 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5812 {
5813     \bool_set_true:N \l_@@_nocolor_used_bool
5814     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5815     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5816     \clist_map_inline:Nn \l_@@_rows_tl
5817     {

```

```

5818     \clist_map_inline:Nn \l_@@_cols_tl
5819     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5820   }
5821 }

```

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 `\l_@@_cols_tl` will be replaced by `2,4,5,6,8,9,10`.

```

5822 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5823 {
5824   \clist_set_eq:NN \l_tmpa_clist #1
5825   \clist_clear:N #1
5826   \clist_map_inline:Nn \l_tmpa_clist
5827   {
5828     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5829     \tl_if_in:NnTF \l_tmpa_tl { - }
5830     { \@@_cut_on_hyphen:w ##1 \q_stop }
5831     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5832     \bool_lazy_or:nnT
5833     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5834     { \tl_if_blank_p:o \l_tmpa_tl }
5835     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5836     \bool_lazy_or:nnT
5837     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5838     { \tl_if_blank_p:o \l_tmpb_tl }
5839     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5840     \int_compare:nNnT \l_tmpb_tl > #2
5841     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5842     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5843     { \clist_put_right:Nn #1 { ####1 } }
5844   }
5845 }

```

The following command will be linked to `\cellcolor` in the `tabular`.

```

5846 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5847 {
5848   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5849   {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```

5850     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5851     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5852   }
5853   \ignorespaces
5854 }

```

The following command will be linked to `\rowcolor` in the `tabular`.

```

5855 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5856 {
5857   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5858   {
5859     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5860     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5861     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5862   }
5863   \ignorespaces
5864 }

```

The following command will be linked to `\rowcolors` in the `tabular`. The last argument (an optional argument between square brackets is taken by currying).

```

5865 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5866 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5867 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5868 {
5869   \peek_remove_spaces:n
5870   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5871 }

5872 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5873 {

```

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`.

```

5874   \seq_gclear:N \g_tmpa_seq
5875   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5876   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5877   \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`.

```

5878   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5879   {
5880     { \int_use:N \c@iRow }
5881     { \exp_not:n { #1 } }
5882     { \exp_not:n { #2 } }
5883     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5884   }
5885 }

```

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`).

```

5886 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5887 {
5888   \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).

```

5889   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5890   {
5891     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5892     {
5893       \@@_rowlistcolors
5894       [ \exp_not:n { #2 } ]
5895       { #1 - \int_eval:n { \c@iRow - 1 } }
5896       { \exp_not:n { #3 } }
5897       [ \exp_not:n { #4 } ]
5898     }
5899   }
5900 }

```

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.

```

5901 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5902 {
5903   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5904     { \@@_rowlistcolors_tabular_ii:nnnn #1 }
5905   \seq_gclear:N \g_@@_rowlistcolors_seq
5906 }

5907 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5908 {
5909   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5910     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5911 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is written in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5912 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5913 {

```

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).

```

5914   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5915   {

```

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).

```

5916     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5917     {
5918       \exp_not:N \columncolor [ #1 ]
5919       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5920     }
5921   }
5922 }

5923 \hook_gput_code:nnn { begindocument } { . }
5924 {
5925   \IfPackageLoadedTF { colortbl }
5926   {
5927     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5928     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5929     \cs_new_protected:Npn \@@_revert_colortbl:
5930     {
5931       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5932       {
5933         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5934         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5935       }
5936     }
5937   }
5938   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5939 }

```

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`.

```
5940 \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`.

```
5941 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5942 {
5943   \int_if_zero:nTF \l_@@_first_col_int
5944     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5945     {
5946       \int_if_zero:nTF \c@jCol
5947         {
5948           \int_compare:nNnF \c@iRow = { -1 }
5949             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5950         }
5951         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5952     }
5953 }
```

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.

```
5954 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5955 {
5956   \int_if_zero:nF \c@iRow
5957   {
5958     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5959     {
5960       \int_compare:nNnT \c@jCol > \c_zero_int
5961       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5962     }
5963   }
5964 }
```

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.

```
5965 \keys_define:nn { nicematrix / Rules }
5966 {
5967   position .int_set:N = \l_@@_position_int ,
5968   position .value_required:n = true ,
5969   start .int_set:N = \l_@@_start_int ,
```

```

5970     end .code:n =
5971     \bool_lazy_or:nnTF
5972     { \tl_if_empty_p:n { #1 } }
5973     { \str_if_eq_p:ee { #1 } { last } }
5974     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5975     { \int_set:Nn \l_@@_end_int { #1 } }
5976 }

```

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.

```

5977 \keys_define:nn { nicematrix / RulesBis }
5978 {
5979     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5980     multiplicity .initial:n = 1 ,
5981     dotted .bool_set:N = \l_@@_dotted_bool ,
5982     dotted .initial:n = false ,
5983     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]`.

```

5984     color .code:n =
5985     \@@_set_CT@arc@:n { #1 }
5986     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5987     color .value_required:n = true ,
5988     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5989     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.

```

5990     tikz .code:n =
5991     \IfPackageLoadedTF { tikz }
5992     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5993     { \@@_error:n { tikz-without-tikz } } ,
5994     tikz .value_required:n = true ,
5995     total-width .dim_set:N = \l_@@_rule_width_dim ,
5996     total-width .value_required:n = true ,
5997     width .meta:n = { total-width = #1 } ,
5998     unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5999 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6000 \cs_new_protected:Npn \@@_vline:n #1
6001 {

```

The group is for the options.

```

6002     \group_begin:
6003     \int_set_eq:NN \l_@@_end_int \c@iRow
6004     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6005     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6006     \@@_vline_i:
6007     \group_end:
6008 }

```

```

6009 \cs_new_protected:Npn \@@_vline_i:
6010 {

```

\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.

```

6011     \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6012     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6013         \l_tmpa_tl
6014     {

```

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.

```

6015         \bool_gset_true:N \g_tmpa_bool
6016         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6017             { \@@_test_vline_in_block:nnnn ##1 }
6018         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6019             { \@@_test_vline_in_block:nnnn ##1 }
6020         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6021             { \@@_test_vline_in_stroken_block:nmmn ##1 }
6022         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6023         \bool_if:NTF \g_tmpa_bool
6024             {
6025                 \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.

```

6026                 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6027             }
6028             {
6029                 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6030                 {
6031                     \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6032                     \@@_vline_ii:
6033                     \int_zero:N \l_@@_local_start_int
6034                 }
6035             }
6036         }
6037         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6038         {
6039             \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6040             \@@_vline_ii:
6041         }
6042     }

```

```

6043 \cs_new_protected:Npn \@@_test_in_corner_v:
6044 {
6045     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6046     {
6047         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6048         { \bool_set_false:N \g_tmpa_bool }
6049     }
6050     {
6051         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6052         {
6053             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6054             { \bool_set_false:N \g_tmpa_bool }
6055         }
6056         \@@_if_in_corner:nT
6057             { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6058             { \bool_set_false:N \g_tmpa_bool }
6059     }

```

```

6060     }
6061   }
6062 }

6063 \cs_new_protected:Npn \@@_vline_ii:
6064 {
6065   \tl_clear:N \l_@@_tikz_rule_tl
6066   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6067   \bool_if:NTF \l_@@_dotted_bool
6068     \@@_vline_iv:
6069     {
6070       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6071         \@@_vline_iii:
6072         \@@_vline_v:
6073     }
6074 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6075 \cs_new_protected:Npn \@@_vline_iii:
6076 {
6077   \pgfpicture
6078   \pgfrememberpicturepositiononpagetrue
6079   \pgf@relevantforpicturesizefalse
6080   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6081   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6082   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6083   \dim_set:Nn \l_tmpb_dim
6084     {
6085     \pgf@x
6086     - 0.5 \l_@@_rule_width_dim
6087     +
6088     ( \arrayrulewidth * \l_@@_multiplicity_int
6089     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6090     }
6091   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6092   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6093   \bool_lazy_all:nT
6094     {
6095     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6096     { \cs_if_exist_p:N \CT@drsc@ }
6097     { ! \tl_if_blank_p:o \CT@drsc@ }
6098     }
6099     {
6100     \group_begin:
6101     \CT@drsc@
6102     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6103     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6104     \dim_set:Nn \l_@@_tmpd_dim
6105       {
6106       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6107       * ( \l_@@_multiplicity_int - 1 )
6108       }
6109     \pgfpathrectanglecorners
6110       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6111       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6112     \pgfusepath { fill }
6113     \group_end:
6114     }
6115   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6116   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6117   \prg_replicate:mn { \l_@@_multiplicity_int - 1 }
6118     {

```

```

6119     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6120     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6121     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6122     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6123   }
6124   \CT@arc@
6125   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6126   \pgfsetrectcap
6127   \pgfusepathqstroke
6128   \endpgfpicture
6129 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6130 \cs_new_protected:Npn \@@_vline_iv:
6131 {
6132   \pgfpicture
6133   \pgfrememberpicturepositiononpagetrue
6134   \pgf@relevantforpicturesizefalse
6135   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6136   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6137   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6138   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6139   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6140   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6141   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6142   \CT@arc@
6143   \@@_draw_line:
6144   \endpgfpicture
6145 }

```

The following code is for the case when the user uses the key `tikz`.

```

6146 \cs_new_protected:Npn \@@_vline_v:
6147 {
6148   \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).

```

6149   \CT@arc@
6150   \tl_if_empty:NF \l_@@_rule_color_tl
6151   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6152   \pgfrememberpicturepositiononpagetrue
6153   \pgf@relevantforpicturesizefalse
6154   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6155   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6156   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6157   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6158   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6159   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6160   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6161   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6162     ( \l_tmpb_dim , \l_tmpa_dim ) --
6163     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6164   \end {tikzpicture }
6165 }

```

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).

```

6166 \cs_new_protected:Npn \@@_draw_vlines:
6167 {
6168   \int_step_inline:nnn
6169     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }

```

```

6170 {
6171   \bool_lazy_or:nntF \g_@@_delims_bool \l_@@_except_borders_bool
6172   \c@jCol
6173   { \int_eval:n { \c@jCol + 1 } }
6174 }
6175 {
6176   \str_if_eq:eeF \l_@@_vlines_clist { all }
6177   { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6178   { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6179 }
6180 }

```

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}`.

```

6181 \cs_new_protected:Npn \@@_hline:n #1
6182 {

```

The group is for the options.

```

6183   \group_begin:
6184   \int_zero_new:N \l_@@_end_int
6185   \int_set_eq:NN \l_@@_end_int \c@jCol
6186   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6187   \@@_hline_i:
6188   \group_end:
6189 }

6190 \cs_new_protected:Npn \@@_hline_i:
6191 {
6192   \int_zero_new:N \l_@@_local_start_int
6193   \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`.

```

6194   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6195   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6196   \l_tmpb_tl
6197   {

```

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.

```

6198   \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6199   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6200   { \@@_test_hline_in_block:nnnn ##1 }

6201   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6202   { \@@_test_hline_in_block:nnnn ##1 }

6203   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6204   { \@@_test_hline_in_stroken_block:nnnn ##1 }

6205   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6206   \bool_if:NTF \g_tmpa_bool
6207   {
6208     \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.

```

6209     { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6210   }
6211   {

```

```

6212         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6213         {
6214             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6215             \@@_hline_ii:
6216             \int_zero:N \l_@@_local_start_int
6217         }
6218     }
6219 }
6220 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6221 {
6222     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6223     \@@_hline_ii:
6224 }
6225 }

6226 \cs_new_protected:Npn \@@_test_in_corner_h:
6227 {
6228     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6229     {
6230         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6231         { \bool_set_false:N \g_tmpa_bool }
6232     }
6233     {
6234         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6235         {
6236             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6237             { \bool_set_false:N \g_tmpa_bool }
6238             {
6239                 \@@_if_in_corner:nT
6240                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6241                 { \bool_set_false:N \g_tmpa_bool }
6242             }
6243         }
6244     }
6245 }

6246 \cs_new_protected:Npn \@@_hline_ii:
6247 {
6248     \tl_clear:N \l_@@_tikz_rule_tl
6249     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6250     \bool_if:NTF \l_@@_dotted_bool
6251     \@@_hline_iv:
6252     {
6253         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6254         \@@_hline_iii:
6255         \@@_hline_v:
6256     }
6257 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6258 \cs_new_protected:Npn \@@_hline_iii:
6259 {
6260     \pgfpicture
6261     \pgfrememberpicturepositiononpagetrue
6262     \pgf@relevantforpicturesizefalse
6263     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6264     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6265     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6266     \dim_set:Nn \l_tmpb_dim
6267     {
6268         \pgf@y

```

```

6269     - 0.5 \l_@@_rule_width_dim
6270     +
6271     ( \arrayrulewidth * \l_@@_multiplicity_int
6272       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6273   }
6274   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6275   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6276   \bool_lazy_all:nT
6277     {
6278       { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6279       { \cs_if_exist_p:N \CT@drsc@ }
6280       { ! \tl_if_blank_p:o \CT@drsc@ }
6281     }
6282     {
6283       \group_begin:
6284       \CT@drsc@
6285       \dim_set:Nn \l_@@_tmpd_dim
6286         {
6287           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6288           * ( \l_@@_multiplicity_int - 1 )
6289         }
6290       \pgfpathrectanglecorners
6291         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6292         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6293       \pgfusepathqfill
6294       \group_end:
6295     }
6296   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6297   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6298   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6299     {
6300       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6301       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6302       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6303       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6304     }
6305   \CT@arc@
6306   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6307   \pgfsetrectcap
6308   \pgfusepathqstroke
6309   \endpgfpicture
6310 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6311 \cs_new_protected:Npn \@@_hline_iv:

```

```

6312 {
6313   \pgfpicture
6314   \pgfrememberpicturepositiononpagetrue
6315   \pgf@relevantforpicturesizefalse
6316   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6317   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6318   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6319   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6320   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6321   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6322     {
6323       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6324       \bool_if:NF \g_@@_delims_bool
6325       { \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.

```

6326     \tl_if_eq:NnF \g_@@_left_delim_tl (
6327       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6328     )
6329   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6330   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6331   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6332     {
6333       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6334       \bool_if:NF \g_@@_delims_bool
6335       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6336       \tl_if_eq:NnF \g_@@_right_delim_tl )
6337       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6338     }
6339   \CT@arc@
6340   \@@_draw_line:
6341   \endpgfpicture
6342 }

```

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`).

```

6343 \cs_new_protected:Npn \@@_hline_v:
6344 {
6345   \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).

```

6346   \CT@arc@
6347   \tl_if_empty:NF \l_@@_rule_color_tl
6348     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6349   \pgfrememberpicturepositiononpagetrue
6350   \pgf@relevantforpicturesizefalse
6351   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6352   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6353   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6354   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6355   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6356   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6357   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6358   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6359     ( \l_tmpa_dim , \l_tmpb_dim ) --
6360     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6361   \end { tikzpicture }
6362 }

```

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).

```

6363 \cs_new_protected:Npn \@@_draw_hlines:
6364 {
6365   \int_step_inline:nnn
6366     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6367     {
6368       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6369         \c@iRow
6370         { \int_eval:n { \c@iRow + 1 } }
6371     }
6372     {
6373       \str_if_eq:eeF \l_@@_hlines_clist { all }
6374       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6375       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6376     }
6377 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6378 \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.

```

6379 \cs_set:Npn \@@_Hline_i:n #1
6380 {
6381   \peek_remove_spaces:n
6382   {
6383     \peek_meaning:NTF \Hline
6384     { \@@_Hline_ii:nn { #1 + 1 } }
6385     { \@@_Hline_iii:n { #1 } }
6386   }
6387 }
6388 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6389 \cs_set:Npn \@@_Hline_iii:n #1
6390 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6391 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6392 {
6393   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6394   \skip_vertical:N \l_@@_rule_width_dim
6395   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6396   {
6397     \@@_hline:n
6398     {
6399       multiplicity = #1 ,
6400       position = \int_eval:n { \c@iRow + 1 } ,
6401       total-width = \dim_use:N \l_@@_rule_width_dim ,
6402       #2
6403     }
6404   }
6405   \egroup
6406 }

```

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`).

```

6407 \cs_new_protected:Npn \@@_custom_line:n #1
6408 {

```

```

6409 \str_clear_new:N \l_@@_command_str
6410 \str_clear_new:N \l_@@_ccommand_str
6411 \str_clear_new:N \l_@@_letter_str
6412 \tl_clear_new:N \l_@@_other_keys_tl
6413 \keys_set:known:nN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6414 \bool_lazy_all:nTF
6415 {
6416   { \str_if_empty_p:N \l_@@_letter_str }
6417   { \str_if_empty_p:N \l_@@_command_str }
6418   { \str_if_empty_p:N \l_@@_ccommand_str }
6419 }
6420 { \@@_error:n { No~letter~and~no~command } }
6421 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6422 }

6423 \keys_define:nn { nicematrix / custom-line }
6424 {
6425   letter .str_set:N = \l_@@_letter_str ,
6426   letter .value_required:n = true ,
6427   command .str_set:N = \l_@@_command_str ,
6428   command .value_required:n = true ,
6429   ccommand .str_set:N = \l_@@_ccommand_str ,
6430   ccommand .value_required:n = true ,
6431 }

6432 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6433 \cs_new_protected:Npn \@@_custom_line_i:n #1
6434 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6435 \bool_set_false:N \l_@@_tikz_rule_bool
6436 \bool_set_false:N \l_@@_dotted_rule_bool
6437 \bool_set_false:N \l_@@_color_bool

6438 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6439 \bool_if:NT \l_@@_tikz_rule_bool
6440 {
6441   \IfPackageLoadedF { tikz }
6442   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6443   \bool_if:NT \l_@@_color_bool
6444   { \@@_error:n { color~in~custom~line~with~tikz } }
6445 }
6446 \bool_if:NT \l_@@_dotted_rule_bool
6447 {
6448   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6449   { \@@_error:n { key~multiplicity~with~dotted } }
6450 }
6451 \str_if_empty:NF \l_@@_letter_str
6452 {
6453   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6454   { \@@_error:n { Several~letters } }
6455   {
6456     \tl_if_in:NoTF
6457     \c_@@_forbidden_letters_str
6458     \l_@@_letter_str
6459     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6460     {

```

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.

```

6461         \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6462         { \@@_v_custom_line:n { #1 } }
6463     }
6464 }
6465 }
6466 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6467 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6468 }
6469 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6470 \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.

```

6471 \keys_define:nn { nicematrix / custom-line-bis }
6472 {
6473     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6474     multiplicity .initial:n = 1 ,
6475     multiplicity .value_required:n = true ,
6476     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6477     color .value_required:n = true ,
6478     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6479     tikz .value_required:n = true ,
6480     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6481     dotted .value_forbidden:n = true ,
6482     total-width .code:n = { } ,
6483     total-width .value_required:n = true ,
6484     width .code:n = { } ,
6485     width .value_required:n = true ,
6486     sep-color .code:n = { } ,
6487     sep-color .value_required:n = true ,
6488     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6489 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6490 \bool_new:N \l_@@_dotted_rule_bool
6491 \bool_new:N \l_@@_tikz_rule_bool
6492 \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`.

```

6493 \keys_define:nn { nicematrix / custom-line-width }
6494 {
6495     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6496     multiplicity .initial:n = 1 ,
6497     multiplicity .value_required:n = true ,
6498     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6499     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6500         \bool_set_true:N \l_@@_total_width_bool ,
6501     total-width .value_required:n = true ,
6502     width .meta:n = { total-width = #1 } ,
6503     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6504 }

```

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`).

```
6505 \cs_new_protected:Npn \@@_h_custom_line:n #1
6506 {
```

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`).

```
6507   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6508   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6509 }
```

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`).

```
6510 \cs_new_protected:Npn \@@_c_custom_line:n #1
6511 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6512   \exp_args:Nc \NewExpandableDocumentCommand
6513     { nicematrix - \l_@@_ccommand_str }
6514     { 0 { } m }
6515     {
6516       \noalign
6517       {
6518         \@@_compute_rule_width:n { #1 , ##1 }
6519         \skip_vertical:n { \l_@@_rule_width_dim }
6520         \clist_map_inline:nn
6521           { ##2 }
6522           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6523       }
6524     }
6525   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6526 }
```

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*.

```
6527 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6528 {
6529   \tl_if_in:nnTF { #2 } { - }
6530     { \@@_cut_on_hyphen:w #2 \q_stop }
6531     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6532   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6533     {
6534       \@@_hline:n
6535       {
6536         #1 ,
6537         start = \l_tmpa_tl ,
6538         end = \l_tmpb_tl ,
6539         position = \int_eval:n { \c@iRow + 1 } ,
6540         total-width = \dim_use:N \l_@@_rule_width_dim
6541       }
6542     }
6543 }
```

```
6544 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6545 {
6546   \bool_set_false:N \l_@@_tikz_rule_bool
6547   \bool_set_false:N \l_@@_total_width_bool
6548   \bool_set_false:N \l_@@_dotted_rule_bool
6549   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6550   \bool_if:NF \l_@@_total_width_bool
```

```

6551 {
6552   \bool_if:NTF \l_@@_dotted_rule_bool
6553   { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6554   {
6555     \bool_if:NF \l_@@_tikz_rule_bool
6556     {
6557       \dim_set:Nn \l_@@_rule_width_dim
6558       {
6559         \arrayrulewidth * \l_@@_multiplicity_int
6560         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6561       }
6562     }
6563   }
6564 }
6565 }

6566 \cs_new_protected:Npn \@@_v_custom_line:n #1
6567 {
6568   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6569   \tl_gput_right:Ne \g_@@_array_preamble_tl
6570   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6571   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6572   {
6573     \@@_vline:n
6574     {
6575       #1 ,
6576       position = \int_eval:n { \c@jCol + 1 } ,
6577       total-width = \dim_use:N \l_@@_rule_width_dim
6578     }
6579   }
6580   \@@_rec_preamble:n
6581 }

6582 \@@_custom_line:n
6583 { 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.

```

6584 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6585 {
6586   \int_compare:nNnT \l_tmpa_tl > { #1 }
6587   {
6588     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6589     {
6590       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6591       {
6592         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6593         { \bool_gset_false:N \g_tmpa_bool }
6594       }
6595     }
6596   }
6597 }

```

The same for vertical rules.

```

6598 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6599 {
6600   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6601   {
6602     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }

```

```

6603     {
6604     \int_compare:nNnT \l_tmpb_tl > { #2 }
6605     {
6606     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6607     { \bool_gset_false:N \g_tmpa_bool }
6608     }
6609     }
6610 }
6611 }
6612 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6613 {
6614 \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6615 {
6616 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6617 {
6618 \int_compare:nNnTF \l_tmpa_tl = { #1 }
6619 { \bool_gset_false:N \g_tmpa_bool }
6620 {
6621 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6622 { \bool_gset_false:N \g_tmpa_bool }
6623 }
6624 }
6625 }
6626 }
6627 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6628 {
6629 \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6630 {
6631 \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6632 {
6633 \int_compare:nNnTF \l_tmpb_tl = { #2 }
6634 { \bool_gset_false:N \g_tmpa_bool }
6635 {
6636 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6637 { \bool_gset_false:N \g_tmpa_bool }
6638 }
6639 }
6640 }
6641 }

```

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.

```

6642 \cs_new_protected:Npn \@@_compute_corners:
6643 {
6644 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6645 { \@@_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`).

```

6646 \clist_clear:N \l_@@_corners_cells_clist
6647 \clist_map_inline:Nn \l_@@_corners_clist
6648 {
6649 \str_case:nnF { ##1 }
6650 {

```

```

6651     { NW }
6652     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6653     { NE }
6654     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6655     { SW }
6656     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6657     { SE }
6658     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6659   }
6660   { \@@_error:nn { bad~corner } { ##1 } }
6661 }

```

Even if the user has used the key corners the list of cells in the corners may be empty.

```

6662   \clist_if_empty:NF \l_@@_corners_cells_clist
6663   {

```

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.

```

6664     \tl_gput_right:Ne \g_@@_aux_tl
6665     {
6666         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6667         { \l_@@_corners_cells_clist }
6668     }
6669 }
6670 }

```

```

6671 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6672 {
6673   \int_step_inline:nnn { #1 } { #3 }
6674   {
6675     \int_step_inline:nnn { #2 } { #4 }
6676     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6677   }
6678 }

```

```

6679 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6680 {
6681   \cs_if_exist:cTF
6682   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6683   \prg_return_true:
6684   \prg_return_false:
6685 }

```

“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.

```

6686 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6687 {

```

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.

```

6688 \bool_set_false:N \l_tmpa_bool
6689 \int_zero_new:N \l_@@_last_empty_row_int
6690 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6691 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6692 {
6693   \bool_lazy_or:nnTF
6694   {
6695     \cs_if_exist_p:c
6696     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6697   }
6698   { \@@_if_in_block_p:nn { ##1 } { #2 } }
6699   { \bool_set_true:N \l_tmpa_bool }
6700   {
6701     \bool_if:NF \l_tmpa_bool
6702     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6703   }
6704 }

```

Now, you determine the last empty cell in the row of number 1.

```

6705 \bool_set_false:N \l_tmpa_bool
6706 \int_zero_new:N \l_@@_last_empty_column_int
6707 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6708 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6709 {
6710   \bool_lazy_or:nnTF
6711   {
6712     \cs_if_exist_p:c
6713     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6714   }
6715   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6716   { \bool_set_true:N \l_tmpa_bool }
6717   {
6718     \bool_if:NF \l_tmpa_bool
6719     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6720   }
6721 }

```

Now, we loop over the rows.

```

6722 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6723 {

```

We treat the row number `##1` with another loop.

```

6724 \bool_set_false:N \l_tmpa_bool
6725 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6726 {
6727   \bool_lazy_or:nnTF
6728   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - ##### } }
6729   { \@@_if_in_block_p:nn { ##1 } { ##### } }
6730   { \bool_set_true:N \l_tmpa_bool }
6731   {
6732     \bool_if:NF \l_tmpa_bool
6733     {
6734       \int_set:Nn \l_@@_last_empty_column_int { ##### }
6735       \clist_put_right:Nn
6736       \l_@@_corners_cells_clist
6737       { ##1 - ##### }
6738       \cs_set_nopar:cpn { @@ _ corner _ ##1 - ##### } { }
6739     }
6740   }

```

```

6741     }
6742   }
6743 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6744 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6745 \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.

```

6746 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6747 \keys_define:nn { nicematrix / NiceMatrixBlock }
6748 {
6749   auto-columns-width .code:n =
6750   {
6751     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6752     \dim_gzero_new:N \g_@@_max_cell_width_dim
6753     \bool_set_true:N \l_@@_auto_columns_width_bool
6754   }
6755 }

6756 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6757 {
6758   \int_gincr:N \g_@@_NiceMatrixBlock_int
6759   \dim_zero:N \l_@@_columns_width_dim
6760   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6761   \bool_if:NT \l_@@_block_auto_columns_width_bool
6762   {
6763     \cs_if_exist:cT
6764     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6765     {
6766       \dim_set:Nn \l_@@_columns_width_dim
6767       {
6768         \use:c
6769         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6770       }
6771     }
6772   }
6773 }

```

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`).

```

6774 {
6775   \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.

```

6776     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6777     {
6778       \bool_if:NT \l_@@_block_auto_columns_width_bool
6779       {
6780         \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6781         \iow_shipout:Ne \@mainaux
6782         {
6783           \cs_gset:cpn
6784           { @@ _ 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.

```

6785           { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6786         }
6787         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6788       }
6789     }
6790     \ignorespacesafterend
6791   }

```

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).

```

6792 \cs_new_protected:Npn \@@_create_extra_nodes:
6793 {
6794   \bool_if:nTF \l_@@_medium_nodes_bool
6795   {
6796     \bool_if:NTF \l_@@_large_nodes_bool
6797     \@@_create_medium_and_large_nodes:
6798     \@@_create_medium_nodes:
6799   }
6800   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6801 }

```

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`.

```

6802 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6803 {
6804   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:

```

```

6805 {
6806   \dim_zero_new:c { l_@@_row_@@_i: _min_dim }
6807   \dim_set_eq:cN { l_@@_row_@@_i: _min_dim } \c_max_dim
6808   \dim_zero_new:c { l_@@_row_@@_i: _max_dim }
6809   \dim_set:cn { l_@@_row_@@_i: _max_dim } { - \c_max_dim }
6810 }
6811 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:
6812 {
6813   \dim_zero_new:c { l_@@_column_@@_j: _min_dim }
6814   \dim_set_eq:cN { l_@@_column_@@_j: _min_dim } \c_max_dim
6815   \dim_zero_new:c { l_@@_column_@@_j: _max_dim }
6816   \dim_set:cn { l_@@_column_@@_j: _max_dim } { - \c_max_dim }
6817 }

```

We begin the two nested loops over the rows and the columns of the array.

```

6818 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
6819 {
6820   \int_step_variable:nnNn
6821     \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.

```

6822 {
6823   \cs_if_exist:cT
6824     { \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`.

```

6825 {
6826   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6827   \dim_set:cn { l_@@_row_@@_i: _min_dim }
6828   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6829   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6830   {
6831     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6832     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6833   }

```

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`.

```

6834   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6835   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6836   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6837   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6838   {
6839     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6840     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6841   }
6842 }
6843 }
6844 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6845 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int @@_i:
6846 {
6847   \dim_compare:nNnT
6848     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6849     {
6850       \@@_qpoint:n { row - \@@_i: - base }
6851       \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6852       \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6853     }
6854 }
6855 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int @@_j:

```

```

6856 {
6857   \dim_compare:nNnT
6858     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6859     {
6860       \@@_qpoint:n { col - \@@_j: }
6861       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6862       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6863     }
6864 }
6865 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6866 \cs_new_protected:Npn \@@_create_medium_nodes:
6867 {
6868   \pgfpicture
6869   \pgfrememberpicturepositiononpagetrue
6870   \pgf@relevantforpicturesizefalse
6871   \@@_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”.

```

6872   \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6873   \@@_create_nodes:
6874   \endpgfpicture
6875 }

```

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:`.

```

6876 \cs_new_protected:Npn \@@_create_large_nodes:
6877 {
6878   \pgfpicture
6879   \pgfrememberpicturepositiononpagetrue
6880   \pgf@relevantforpicturesizefalse
6881   \@@_computations_for_medium_nodes:
6882   \@@_computations_for_large_nodes:
6883   \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
6884   \@@_create_nodes:
6885   \endpgfpicture
6886 }

6887 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6888 {
6889   \pgfpicture
6890   \pgfrememberpicturepositiononpagetrue
6891   \pgf@relevantforpicturesizefalse
6892   \@@_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”.

```

6893   \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6894   \@@_create_nodes:
6895   \@@_computations_for_large_nodes:
6896   \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
6897   \@@_create_nodes:
6898   \endpgfpicture
6899 }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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.

```
6900 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6901 {
6902   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6903   \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`.

```
6904   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6905   {
6906     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6907     {
6908       (
6909         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6910         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6911       )
6912       / 2
6913     }
6914     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6915     { l_@@_row _ \@@_i: _ min _ dim }
6916   }
6917   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6918   {
6919     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6920     {
6921       (
6922         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6923         \dim_use:c
6924           { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6925       )
6926       / 2
6927     }
6928     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6929     { l_@@_column _ \@@_j: _ max _ dim }
6930   }
```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```
6931   \dim_sub:cn
6932     { l_@@_column _ 1 _ min _ dim }
6933     \l_@@_left_margin_dim
6934   \dim_add:cn
6935     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6936     \l_@@_right_margin_dim
6937 }
```

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).

```
6938 \cs_new_protected:Npn \@@_create_nodes:
6939 {
6940   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6941   {
6942     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6943     {
```

We draw the rectangular node for the cell $(\@@_i-\@@_j)$.

```
6944       \@@_pgf_rect_node:nnnnn
6945       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6946       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
```

```

6947     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6948     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6949     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6950 \str_if_empty:NF \l_@@_name_str
6951     {
6952       \pgfnodealias
6953       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6954       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6955     }
6956   }
6957 }

```

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 .

```

6958   \seq_map_pairwise_function:NNN
6959   \g_@@_multicolumn_cells_seq
6960   \g_@@_multicolumn_sizes_seq
6961   \@@_node_for_multicolumn:nn
6962 }

6963 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6964 {
6965   \cs_set_nopar:Npn \@@_i: { #1 }
6966   \cs_set_nopar:Npn \@@_j: { #2 }
6967 }

```

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”).

```

6968 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6969 {
6970   \@@_extract_coords_values: #1 \q_stop
6971   \@@_pgf_rect_node:nnnnn
6972   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6973   { \dim_use:c { l_@@_column_ \@@_j: _min _ dim } }
6974   { \dim_use:c { l_@@_row_ \@@_i: _min _ dim } }
6975   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6976   { \dim_use:c { l_@@_row_ \@@_i: _ max _ dim } }
6977   \str_if_empty:NF \l_@@_name_str
6978   {
6979     \pgfnodealias
6980     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6981     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6982   }
6983 }

```

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).

```

6984 \keys_define:nn { nicematrix / Block / FirstPass }
6985 {
6986   j .code:n = \str_set:Nn \l_@@_hpos_block_str j

```

```

6987         \bool_set_true:N \l_@@_p_block_bool ,
6988     j .value_forbidden:n = true ,
6989     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6990     l .value_forbidden:n = true ,
6991     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6992     r .value_forbidden:n = true ,
6993     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6994     c .value_forbidden:n = true ,
6995     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6996     L .value_forbidden:n = true ,
6997     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6998     R .value_forbidden:n = true ,
6999     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7000     C .value_forbidden:n = true ,
7001     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7002     t .value_forbidden:n = true ,
7003     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7004     T .value_forbidden:n = true ,
7005     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7006     b .value_forbidden:n = true ,
7007     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7008     B .value_forbidden:n = true ,
7009     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7010     m .value_forbidden:n = true ,
7011     v-center .meta:n = m ,
7012     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7013     p .value_forbidden:n = true ,
7014     color .code:n =
7015         \@@_color:n { #1 }
7016         \tl_set_rescan:Nnn
7017             \l_@@_draw_tl
7018             { \char_set_catcode_other:N ! }
7019             { #1 } ,
7020     color .value_required:n = true ,
7021     respect-arraystretch .code:n =
7022         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7023     respect-arraystretch .value_forbidden:n = true ,
7024 }

```

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.

```

7025 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7026 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7027 {

```

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).

```

7028     \peek_remove_spaces:n
7029     {
7030         \tl_if_blank:nTF { #2 }
7031             { \@@_Block_ii:nmmm \c_one_int \c_one_int }
7032             {
7033                 \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7034                 \@@_Block_i_czech \@@_Block_i
7035                 #2 \q_stop
7036             }
7037         { #1 } { #3 } { #4 }
7038     }
7039 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7040 \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`.

```
7041 {
7042   \char_set_catcode_active:N -
7043   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7044 }
```

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.

```
7045 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7046 {
```

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).

```
7047   \bool_lazy_or:nnTF
7048     { \tl_if_blank_p:n { #1 } }
7049     { \str_if_eq_p:ee { * } { #1 } }
7050     { \int_set:Nn \l_tmpa_int { 100 } }
7051     { \int_set:Nn \l_tmpa_int { #1 } }
7052   \bool_lazy_or:nnTF
7053     { \tl_if_blank_p:n { #2 } }
7054     { \str_if_eq_p:ee { * } { #2 } }
7055     { \int_set:Nn \l_tmpb_int { 100 } }
7056     { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7057   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7058     {
7059       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7060         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7061         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7062     }
7063     { \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.

```
7064   \keys_set:known:nn { nicematrix / Block / FirstPass } { #3 }
7065   \tl_set:Ne \l_tmpa_tl
7066   {
7067     { \int_use:N \c@iRow }
7068     { \int_use:N \c@jCol }
7069     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7070     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7071   }
```

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).

```
7072   \bool_set_false:N \l_tmpa_bool
7073   \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7074     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7075   \bool_case:nF
7076     {
7077       \l_tmpa_bool                { \@@_Block_vii:eennn }
7078       \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.

```

7079     \l_@@_X_bool                { \@@_Block_v:eennn }
7080     { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7081     { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7082     { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7083   }
7084   { \@@_Block_v:eennn }
7085   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7086 }

```

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:nnnnn` 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.

```

7087 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7088 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7089   {
7090     \int_gincr:N \g_@@_block_box_int
7091     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7092       {
7093         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7094           {
7095             \@@_actually_diagbox:nnnnn
7096             { \int_use:N \c@iRow }
7097             { \int_use:N \c@jCol }
7098             { \int_eval:n { \c@iRow + #1 - 1 } }
7099             { \int_eval:n { \c@jCol + #2 - 1 } }
7100             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7101             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7102           }
7103       }
7104     \box_gclear_new:c
7105     { 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!

```

7106   \hbox_gset:cn
7107   { g_@@_block_box_int _ \int_use:N \g_@@_block_box_int _ box }
7108   {

```

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}`).

```

7109      \tl_if_empty:NTF \l_@@_color_tl
7110      { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7111      { \@@_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`.

```

7112      \int_compare:nNnT { #1 } = \c_one_int
7113      {
7114          \int_if_zero:nTF \c@iRow
7115          {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7116      \cs_set_eq:NN \Block \@@_NullBlock:
7117      \l_@@_code_for_first_row_tl
7118      }
7119      {
7120          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7121          {
7122              \cs_set_eq:NN \Block \@@_NullBlock:
7123              \l_@@_code_for_last_row_tl
7124          }
7125      }
7126      \g_@@_row_style_tl
7127      }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7128      \@@_reset_arraystretch:
7129      \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7130      #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.).

```

7131      \@@_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}`.

```

7132      \bool_if:NTF \l_@@_tabular_bool

```

```

7133     {
7134         \bool_lazy_all:nTF
7135         {
7136             { \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.

```

7137             { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7138             { ! \g_@@_rotate_bool }
7139         }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7140     {
7141         \use:e
7142         {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7143             \exp_not:N \begin { minipage }%
7144             [ \str_lowercase:o \l_@@_vpos_block_str ]
7145             { \l_@@_col_width_dim }
7146             \str_case:on \l_@@_hpos_block_str
7147             { c \centering r \raggedleft l \raggedright }
7148         }
7149         #5
7150     \end { minipage }
7151 }

```

In the other cases, we use a `{tabular}`.

```

7152     {
7153         \bool_if:NT \c_@@_testphase_table_bool
7154         { \tagpdfsetup { table / tagging = presentation } }
7155         \use:e
7156         {
7157             \exp_not:N \begin { tabular }%
7158             [ \str_lowercase:o \l_@@_vpos_block_str ]
7159             { @ { } \l_@@_hpos_block_str @ { } }
7160         }
7161         #5
7162     \end { tabular }
7163 }
7164 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7165     {
7166         \c_math_toggle_token
7167         \use:e
7168         {
7169             \exp_not:N \begin { array }%
7170             [ \str_lowercase:o \l_@@_vpos_block_str ]
7171             { @ { } \l_@@_hpos_block_str @ { } }
7172         }
7173         #5
7174     \end { array }
7175     \c_math_toggle_token
7176 }
7177 }

```

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).

```

7178     \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.

```

7179 \int_compare:nNnT { #2 } = \c_one_int
7180 {
7181   \dim_gset:Nn \g_@@_blocks_wd_dim
7182   {
7183     \dim_max:nn
7184     \g_@@_blocks_wd_dim
7185     {
7186       \box_wd:c
7187       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7188     }
7189   }
7190 }

```

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.

```

7191 \bool_lazy_and:nnT
7192 { \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.

```

7193 { \str_if_empty_p:N \l_@@_vpos_block_str }
7194 {
7195   \dim_gset:Nn \g_@@_blocks_ht_dim
7196   {
7197     \dim_max:nn
7198     \g_@@_blocks_ht_dim
7199     {
7200       \box_ht:c
7201       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7202     }
7203   }
7204   \dim_gset:Nn \g_@@_blocks_dp_dim
7205   {
7206     \dim_max:nn
7207     \g_@@_blocks_dp_dim
7208     {
7209       \box_dp:c
7210       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7211     }
7212   }
7213 }
7214 \seq_gput_right:Ne \g_@@_blocks_seq
7215 {
7216   \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.

```

7217 {
7218   \exp_not:n { #3 } ,
7219   \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7220 \bool_if:NT \g_@@_rotate_bool
7221 {
7222   \bool_if:NTF \g_@@_rotate_c_bool
7223   { m }
7224   { \int_compare:nNnT \c_iRow = \l_@@_last_row_int T }
7225 }
7226 }

```

```

7227     {
7228     \box_use_drop:c
7229     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7230     }
7231   }
7232 \bool_set_false:N \g_@@_rotate_c_bool
7233 }

7234 \cs_new:Npn \@@_adjust_hpos_rotate:
7235 {
7236   \bool_if:NT \g_@@_rotate_bool
7237   {
7238     \str_set:Ne \l_@@_hpos_block_str
7239     {
7240       \bool_if:NTF \g_@@_rotate_c_bool
7241       { c }
7242       {
7243         \str_case:onF \l_@@_vpos_block_str
7244         { b l B l t r T r }
7245         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7246       }
7247     }
7248   }
7249 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7250 \cs_new_protected:Npn \@@_rotate_box_of_block:
7251 {
7252   \box_grotate:cn
7253   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7254   { 90 }
7255   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7256   {
7257     \vbox_gset_top:cn
7258     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7259     {
7260       \skip_vertical:n { 0.8 ex }
7261       \box_use:c
7262       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7263     }
7264   }
7265   \bool_if:NT \g_@@_rotate_c_bool
7266   {
7267     \hbox_gset:cn
7268     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7269     {
7270       \c_math_toggle_token
7271       \vcenter
7272       {
7273         \box_use:c
7274         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7275       }
7276       \c_math_toggle_token
7277     }
7278   }
7279 }

```

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 $key=value$ 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.

```

7280 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7281 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7282 {
7283   \seq_gput_right:Ne \g_@@_blocks_seq
7284   {
7285     \l_tmpa_tl
7286     { \exp_not:n { #3 } }
7287     {
7288       \bool_if:NTF \l_@@_tabular_bool
7289       {
7290         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7291     \@@_reset_arraystretch:
7292     \exp_not:n
7293     {
7294       \dim_zero:N \extrarowheight
7295       #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).

```

7296     \bool_if:NT \c_@@_testphase_table_bool
7297     { \tag_stop:n { table } }
7298     \use:e
7299     {
7300       \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7301       { @ { } \l_@@_hpos_block_str @ { } }
7302     }
7303     #5
7304     \end { tabular }
7305   }
7306   \group_end:
7307 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7308   {
7309     \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7310     \@@_reset_arraystretch:
7311     \exp_not:n
7312     {
7313       \dim_zero:N \extrarowheight
7314       #4
7315       \c_math_toggle_token
7316       \use:e
7317       {
7318         \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7319         { @ { } \l_@@_hpos_block_str @ { } }
7320       }
7321       #5
7322       \end { array }
7323       \c_math_toggle_token
7324     }
7325     \group_end:
7326   }
7327 }
7328 }
7329 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7330 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7331 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7332 {
7333   \seq_gput_right:Ne \g_@@_blocks_seq
7334   {
7335     \l_tmpa_tl
7336     { \exp_not:n { #3 } }
7337     {
7338       \group_begin:
7339       \exp_not:n { #4 #5 }
7340       \group_end:
7341     }
7342   }
7343 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7344 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7345 \cs_new_protected:Npn \@@_Block_vii: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     { \exp_not:n { #4 #5 } }
7352   }
7353 }

```

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).

```

7354 \keys_define:nn { nicematrix / Block / SecondPass }
7355 {
7356   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7357   ampersand-in-blocks .default:n = true ,
7358   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7359   tikz .code:n =
7360     \IfPackageLoadedTF { tikz }
7361     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7362     { \@@_error:n { tikz~key~without~tikz } } ,
7363   tikz .value_required:n = true ,
7364   fill .code:n =
7365     \tl_set_rescan:Nnn
7366     \l_@@_fill_tl
7367     { \char_set_catcode_other:N ! }
7368     { #1 } ,
7369   fill .value_required:n = true ,
7370   opacity .tl_set:N = \l_@@_opacity_tl ,
7371   opacity .value_required:n = true ,
7372   draw .code:n =
7373     \tl_set_rescan:Nnn
7374     \l_@@_draw_tl
7375     { \char_set_catcode_other:N ! }
7376     { #1 } ,
7377   draw .default:n = default ,
7378   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7379   rounded-corners .default:n = 4 pt ,
7380   color .code:n =
7381     \@@_color:n { #1 }

```

```

7382 \tl_set_rescan:Nnn
7383 \l_@@_draw_tl
7384 { \char_set_catcode_other:N ! }
7385 { #1 } ,
7386 borders .clist_set:N = \l_@@_borders_clist ,
7387 borders .value_required:n = true ,
7388 hvlines .meta:n = { vlines , hlines } ,
7389 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7390 vlines .default:n = true ,
7391 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7392 hlines .default:n = true ,
7393 line-width .dim_set:N = \l_@@_line_width_dim ,
7394 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7395 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7396 \bool_set_true:N \l_@@_p_block_bool ,
7397 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7398 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7399 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7400 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7401 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7402 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7403 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7404 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7405 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7406 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7407 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7408 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7409 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7410 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7411 m .value_forbidden:n = true ,
7412 v-center .meta:n = m ,
7413 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7414 p .value_forbidden:n = true ,
7415 name .tl_set:N = \l_@@_block_name_str ,
7416 name .value_required:n = true ,
7417 name .initial:n = ,
7418 respect-arraystretch .code:n =
7419 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7420 respect-arraystretch .value_forbidden:n = true ,
7421 transparent .bool_set:N = \l_@@_transparent_bool ,
7422 transparent .default:n = true ,
7423 transparent .initial:n = false ,
7424 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7425 }

```

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.

```

7426 \cs_new_protected:Npn \@@_draw_blocks:
7427 {
7428 \bool_if:NTF \c_@@_recent_array_bool
7429 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7430 { \cs_set_eq:NN \ialign \@@_old_ialign: }
7431 \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7432 }
7433 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7434 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7435 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7436 \int_zero_new:N \l_@@_last_row_int
7437 \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.

```

7438 \int_compare:nNnTF { #3 } > { 99 }
7439   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7440   { \int_set:Nn \l_@@_last_row_int { #3 } }
7441 \int_compare:nNnTF { #4 } > { 99 }
7442   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7443   { \int_set:Nn \l_@@_last_col_int { #4 } }
7444 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7445   {
7446     \bool_lazy_and:nnTF
7447       \l_@@_preamble_bool
7448       {
7449         \int_compare_p:n
7450           { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7451       }
7452     {
7453       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7454       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7455       \@@_msg_redirect_name:nn { columns-not-used } { none }
7456     }
7457     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7458   }
7459   {
7460     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7461     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7462     {
7463       \@@_Block_v:nneenn
7464       { #1 }
7465       { #2 }
7466       { \int_use:N \l_@@_last_row_int }
7467       { \int_use:N \l_@@_last_col_int }
7468       { #5 }
7469       { #6 }
7470     }
7471   }
7472 }

```

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

```

7473 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7474   {

```

The group is for the keys.

```

7475   \group_begin:
7476   \int_compare:nNnT { #1 } = { #3 }
7477     { \str_set:Nn \l_@@_vpos_block_str { t } }
7478   \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`.

```

7479   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7480   \bool_lazy_and:nnT
7481     \l_@@_vlines_block_bool
7482     { ! \l_@@_ampersand_bool }

```

```

7483 {
7484   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7485   {
7486     \@@_vlines_block:nnn
7487     { \exp_not:n { #5 } }
7488     { #1 - #2 }
7489     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7490   }
7491 }
7492 \bool_if:NT \l_@@_hlines_block_bool
7493 {
7494   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7495   {
7496     \@@_hlines_block:nnn
7497     { \exp_not:n { #5 } }
7498     { #1 - #2 }
7499     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7500   }
7501 }
7502 \bool_if:NF \l_@@_transparent_bool
7503 {
7504   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7505   {

```

The sequence of the positions of the blocks (excepted the blocks with the key hvlines) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7506     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7507     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7508   }
7509 }

7510 \tl_if_empty:NF \l_@@_draw_tl
7511 {
7512   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7513   { \@@_error:n { hlines~with~color } }
7514   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7515   {
7516     \@@_stroke_block:nnn

```

#5 are the options

```

7517     { \exp_not:n { #5 } }
7518     { #1 - #2 }
7519     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7520   }
7521   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7522   { { #1 } { #2 } { #3 } { #4 } }
7523 }

7524 \clist_if_empty:NF \l_@@_borders_clist
7525 {
7526   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7527   {
7528     \@@_stroke_borders_block:nnn
7529     { \exp_not:n { #5 } }
7530     { #1 - #2 }
7531     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7532   }
7533 }

7534 \tl_if_empty:NF \l_@@_fill_tl
7535 {
7536   \@@_add_opacity_to_fill:
7537   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7538   {

```

```

7539         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7540         { #1 - #2 }
7541         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7542         { \dim_use:N \l_@@_rounded_corners_dim }
7543     }
7544 }
7545 \seq_if_empty:NF \l_@@_tikz_seq
7546 {
7547     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7548     {
7549         \@@_block_tikz:nnnnn
7550         { \seq_use:Nn \l_@@_tikz_seq { , } }
7551         { #1 }
7552         { #2 }
7553         { \int_use:N \l_@@_last_row_int }
7554         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7555     }
7556 }
7557 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7558 {
7559     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7560     {
7561         \@@_actually_diagbox:nnnnn
7562         { #1 }
7563         { #2 }
7564         { \int_use:N \l_@@_last_row_int }
7565         { \int_use:N \l_@@_last_col_int }
7566         { \exp_not:n { ##1 } }
7567         { \exp_not:n { ##2 } }
7568     }
7569 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\
& & & two & \\
three & & four & five & \\
six & & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block	one
three	two
six	five
seven	eight

The construction of the node corresponding to the merged cells.

```

7570 \pgfpicture
7571 \pgfrememberpicturepositiononpagetrue
7572 \pgf@relevantforpicturesizefalse
7573 \@@_qpoint:n { row - #1 }
7574 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7575 \@@_qpoint:n { col - #2 }
7576 \dim_set_eq:NN \l_tmpb_dim \pgf@x

```

```

7577 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7578 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7579 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7580 \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.

```

7581 \@@_pgf_rect_node:nnnnn
7582 { \@@_env: - #1 - #2 - block }
7583 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7584 \str_if_empty:NF \l_@@_block_name_str
7585 {
7586   \pgfnodealias
7587   { \@@_env: - \l_@@_block_name_str }
7588   { \@@_env: - #1 - #2 - block }
7589   \str_if_empty:NF \l_@@_name_str
7590   {
7591     \pgfnodealias
7592     { \l_@@_name_str - \l_@@_block_name_str }
7593     { \@@_env: - #1 - #2 - block }
7594   }
7595 }

```

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.

```

7596 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7597 {
7598   \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.

```

7599 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7600 {

```

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.

```

7601   \cs_if_exist:cT
7602   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7603   {
7604     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7605     {
7606       \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7607       \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7608     }
7609   }
7610 }

```

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.

```

7611 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7612 {
7613   \@@_qpoint:n { col - #2 }
7614   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7615 }
7616 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7617 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7618 {
7619   \cs_if_exist:cT
7620   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7621   {
7622     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }

```

```

7623     {
7624         \pgfpointanchor
7625         { \@@_env: - #1 - \int_use:N \l_@@_last_col_int }
7626         { east }
7627         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7628     }
7629 }
7630 }
7631 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7632 {
7633     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7634     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7635 }
7636 \@@_pgf_rect_node:nnnn
7637 { \@@_env: - #1 - #2 - block - short }
7638 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7639 }

```

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.

```

7640 \bool_if:NT \l_@@_medium_nodes_bool
7641 {
7642     \@@_pgf_rect_node:nnn
7643     { \@@_env: - #1 - #2 - block - medium }
7644     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7645     {
7646         \pgfpointanchor
7647         { \@@_env:
7648             - \int_use:N \l_@@_last_row_int
7649             - \int_use:N \l_@@_last_col_int - medium
7650         }
7651         { south-east }
7652     }
7653 }
7654 \endpgfpicture

```

```

7655 \bool_if:NTF \l_@@_ampersand_bool
7656 {
7657     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7658     \int_zero_new:N \l_@@_split_int
7659     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7660     \pgfpicture
7661     \pgfrememberpicturepositiononpagetrue
7662     \pgf@relevantforpicturesizefalse
7663
7664     \@@_qpoint:n { row - #1 }
7665     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7666     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7667     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7668     \@@_qpoint:n { col - #2 }
7669     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7670     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7671     \dim_set:Nn \l_tmpb_dim
7672     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7673     \bool_lazy_or:nnT
7674     \l_@@_vlines_block_bool
7675     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7676     {
7677         \int_step_inline:nn { \l_@@_split_int - 1 }
7678         {
7679             \pgfpathmoveto
7680             {
7681                 \pgfpoint

```

```

7682         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7683         \l_@@_tmpc_dim
7684     }
7685     \pgfpathlineto
7686     {
7687         \pgfpoint
7688         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7689         \l_@@_tmpd_dim
7690     }
7691     \CT@arc@
7692     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7693     \pgfsetrectcap
7694     \pgfusepathqstroke
7695 }
7696 }
7697 \@@_qpoint:n { row - #1 - base }
7698 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7699 \int_step_inline:nn \l_@@_split_int
7700 {
7701     \group_begin:
7702     \dim_set:Nn \col@sep
7703     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7704     \pgftransformshift
7705     {
7706         \pgfpoint
7707         {
7708             \l_tmpa_dim + ##1 \l_tmpb_dim -
7709             \str_case:on \l_@@_hpos_block_str
7710             {
7711                 l { \l_tmpb_dim + \col@sep}
7712                 c { 0.5 \l_tmpb_dim }
7713                 r { \col@sep }
7714             }
7715         }
7716         { \l_@@_tmpc_dim }
7717     }
7718     \pgfset { inner~sep = \c_zero_dim }
7719     \pgfnode
7720     { rectangle }
7721     {
7722         \str_case:on \l_@@_hpos_block_str
7723         {
7724             c { base }
7725             l { base-west }
7726             r { base-east }
7727         }
7728     }
7729     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7730     \group_end:
7731 }
7732 \endpgfpicture
7733 }

```

Now the case where there is no ampersand & in the content of the block.

```

7734 {
7735     \bool_if:NTF \l_@@_p_block_bool
7736     {

```

When the final user has used the key p, we have to compute the width.

```

7737     \pgfpicture
7738     \pgfrememberpicturepositiononpagetrue
7739     \pgf@relevantforpicturesizefalse
7740     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7741     {

```

```

7742         \@@_qpoint:n { col - #2 }
7743         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7744         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7745     }
7746     {
7747         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7748         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7749         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7750     }
7751     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7752 \endpgfpicture
7753 \hbox_set:Nn \l_@@_cell_box
7754 {
7755     \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7756     { \g_tmpb_dim }
7757     \str_case:on \l_@@_hpos_block_str
7758     { c \centering r \raggedleft l \raggedright j { } }
7759     #6
7760     \end { minipage }
7761 }
7762 }
7763 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7764 \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.

```

7765 \pgfpicture
7766 \pgfrememberpicturepositiononpagetrue
7767 \pgf@relevantforpicturesizefalse
7768 \bool_lazy_any:nTF
7769 {
7770     { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7771     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7772     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7773     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7774 }
7775 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7776     \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`.

```

7777     \bool_if:nT \g_@@_last_col_found_bool
7778     {
7779         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7780         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7781     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7782     \tl_set:Ne \l_tmpa_tl
7783     {
7784         \str_case:on \l_@@_vpos_block_str
7785         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7786         { } { % added 2024-06-29
7787             \str_case:on \l_@@_hpos_block_str
7788             {
7789                 c { center }
7790                 l { west }
7791                 r { east }
7792                 j { center }

```

```

7793         }
7794     }
7795     c {
7796         \str_case:on \l_@@_hpos_block_str
7797         {
7798             c { center }
7799             l { west }
7800             r { east }
7801             j { center }
7802         }
7803     }
7804     }
7805     T {
7806         \str_case:on \l_@@_hpos_block_str
7807         {
7808             c { north }
7809             l { north-west }
7810             r { north-east }
7811             j { north }
7812         }
7813     }
7814     }
7815     B {
7816         \str_case:on \l_@@_hpos_block_str
7817         {
7818             c { south }
7819             l { south-west }
7820             r { south-east }
7821             j { south }
7822         }
7823     }
7824     }
7825     }
7826     }
7827     \pgftransformshift
7828     {
7829         \pgfpointanchor
7830         {
7831             \@@_env: - #1 - #2 - block
7832             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7833         }
7834         { \l_tmpa_tl }
7835     }
7836     \pgfset { inner~sep = \c_zero_dim }
7837     \pgfnode
7838     { rectangle }
7839     { \l_tmpa_tl }
7840     { \box_use_drop:N \l_@@_cell_box } { } { }
7841     }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7842     {
7843         \pgfextracty \l_tmpa_dim
7844         {
7845             \@@_qpoint:n
7846             {
7847                 row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7848                 - base
7849             }
7850         }
7851         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

7852     \pgfpointanchor
7853     {
7854         \@@_env: - #1 - #2 - block
7855         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7856     }
7857     {
7858         \str_case:on \l_@@_hpos_block_str
7859         {
7860             c { center }
7861             l { west }
7862             r { east }
7863             j { center }
7864         }
7865     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7866     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7867     \pgfset { inner~sep = \c_zero_dim }
7868     \pgfnode
7869     { rectangle }
7870     {
7871         \str_case:on \l_@@_hpos_block_str
7872         {
7873             c { base }
7874             l { base~west }
7875             r { base~east }
7876             j { base }
7877         }
7878     }
7879     { \box_use_drop:N \l_@@_cell_box } { } { }
7880 }
7881 \endpgfpicture
7882 }
7883 \group_end:
7884 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

7885 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
7886 {
7887     \pgfpicture
7888     \pgfrememberpicturepositiononpagetrue
7889     \pgf@relevantforpicturesizefalse
7890     \pgfpathrectanglecorners
7891     { \pgfpoint { #2 } { #3 } }
7892     { \pgfpoint { #4 } { #5 } }
7893     \pgfsetfillcolor { #1 }
7894     \pgfusepath { fill }
7895     \endpgfpicture
7896 }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

7897 \cs_new_protected:Npn \@@_add_opacity_to_fill:
7898 {
7899     \tl_if_empty:NF \l_@@_opacity_tl
7900     {
7901         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7902         {
7903             \tl_set:Ne \l_@@_fill_tl
7904             {

```

```

7905         [ opacity = \l_@@_opacity_tl ,
7906         \tl_tail:o \l_@@_fill_tl
7907     ]
7908     }
7909     {
7910     \tl_set:Ne \l_@@_fill_tl
7911     { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7912     }
7913 }
7914 }

```

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).

```

7915 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7916 {
7917     \group_begin:
7918     \tl_clear:N \l_@@_draw_tl
7919     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7920     \keys_set_known:n { nicematrix / BlockStroke } { #1 }
7921     \pgfpicture
7922     \pgfrememberpicturepositiononpagetrue
7923     \pgf@relevantforpicturesizefalse
7924     \tl_if_empty:NF \l_@@_draw_tl
7925     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7926     \tl_if_eq:NnTF \l_@@_draw_tl { default }
7927     { \CT@arc@ }
7928     { \@@_color:o \l_@@_draw_tl }
7929     }
7930     \pgfsetcornersarced
7931     {
7932     \pgfpoint
7933     { \l_@@_rounded_corners_dim }
7934     { \l_@@_rounded_corners_dim }
7935     }
7936     \@@_cut_on_hyphen:w #2 \q_stop
7937     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7938     {
7939     \int_compare:nNnF \l_tmpb_tl > \c@jCol
7940     {
7941     \@@_qpoint:n { row - \l_tmpa_tl }
7942     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7943     \@@_qpoint:n { col - \l_tmpb_tl }
7944     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7945     \@@_cut_on_hyphen:w #3 \q_stop
7946     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7947     { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7948     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7949     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7950     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7951     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7952     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7953     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7954     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7955     \pgfpathrectanglecorners
7956     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7957     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7958     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7959     { \pgfusepathqstroke }
7960     { \pgfusepath { stroke } }

```

```

7961     }
7962   }
7963   \endpgfpicture
7964   \group_end:
7965 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7966 \keys_define:nn { nicematrix / BlockStroke }
7967 {
7968   color .tl_set:N = \l_@@_draw_tl ,
7969   draw .code:n =
7970     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7971   draw .default:n = default ,
7972   line-width .dim_set:N = \l_@@_line_width_dim ,
7973   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7974   rounded-corners .default:n = 4 pt
7975 }

```

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).

```

7976 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7977 {
7978   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7979   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7980   \@@_cut_on_hyphen:w #2 \q_stop
7981   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7982   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7983   \@@_cut_on_hyphen:w #3 \q_stop
7984   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7985   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7986   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7987   {
7988     \use:e
7989     {
7990       \@@_vline:n
7991       {
7992         position = ##1 ,
7993         start = \l_@@_tmpc_tl ,
7994         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7995         total-width = \dim_use:N \l_@@_line_width_dim
7996       }
7997     }
7998   }
7999 }
8000 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8001 {
8002   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8003   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8004   \@@_cut_on_hyphen:w #2 \q_stop
8005   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8006   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8007   \@@_cut_on_hyphen:w #3 \q_stop
8008   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8009   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8010   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8011   {
8012     \use:e
8013     {
8014       \@@_hline:n
8015       {
8016         position = ##1 ,
8017         start = \l_@@_tmpd_tl ,

```

```

8018         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8019         total-width = \dim_use:N \l_@@_line_width_dim
8020     }
8021 }
8022 }
8023 }

```

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).

```

8024 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8025 {
8026     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8027     \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8028     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8029     { \@@_error:n { borders~forbidden } }
8030     {
8031         \tl_clear_new:N \l_@@_borders_tikz_tl
8032         \keys_set:no
8033             { nicematrix / OnlyForTikzInBorders }
8034         \l_@@_borders_clist
8035         \@@_cut_on_hyphen:w #2 \q_stop
8036         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8037         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8038         \@@_cut_on_hyphen:w #3 \q_stop
8039         \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8040         \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8041         \@@_stroke_borders_block_i:
8042     }
8043 }
8044 \hook_gput_code:nnn { begindocument } { . }
8045 {
8046     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8047     {
8048         \c_@@_pgfortikzpicture_tl
8049         \@@_stroke_borders_block_ii:
8050         \c_@@_endpgfortikzpicture_tl
8051     }
8052 }
8053 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8054 {
8055     \pgfrememberpicturepositiononpagetrue
8056     \pgf@relevantforpicturesizefalse
8057     \CT@arc@
8058     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8059     \clist_if_in:NnT \l_@@_borders_clist { right }
8060     { \@@_stroke_vertical:n \l_tmpb_tl }
8061     \clist_if_in:NnT \l_@@_borders_clist { left }
8062     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8063     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8064     { \@@_stroke_horizontal:n \l_tmpa_tl }
8065     \clist_if_in:NnT \l_@@_borders_clist { top }
8066     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8067 }
8068 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8069 {
8070     tikz .code:n =
8071         \cs_if_exist:NTF \tikzpicture
8072         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8073         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8074     tikz .value_required:n = true ,
8075     top .code:n = ,

```

```

8076   bottom .code:n = ,
8077   left .code:n = ,
8078   right .code:n = ,
8079   unknown .code:n = \@_error:n { bad~border }
8080 }

```

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).

```

8081 \cs_new_protected:Npn \@_stroke_vertical:n #1
8082 {
8083   \@_qpoint:n \l_@_tmpc_tl
8084   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@_line_width_dim }
8085   \@_qpoint:n \l_tmpa_tl
8086   \dim_set:Nn \l_@_tmpc_dim { \pgf@y + 0.5 \l_@_line_width_dim }
8087   \@_qpoint:n { #1 }
8088   \tl_if_empty:NTF \l_@_borders_tikz_tl
8089   {
8090     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8091     \pgfpathlineto { \pgfpoint \pgf@x \l_@_tmpc_dim }
8092     \pgfusepathqstroke
8093   }
8094   {
8095     \use:e { \exp_not:N \draw [ \l_@_borders_tikz_tl ] }
8096     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@_tmpc_dim ) ;
8097   }
8098 }

```

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).

```

8099 \cs_new_protected:Npn \@_stroke_horizontal:n #1
8100 {
8101   \@_qpoint:n \l_@_tmpd_tl
8102   \clist_if_in:NnTF \l_@_borders_clist { left }
8103   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@_line_width_dim } }
8104   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@_line_width_dim } }
8105   \@_qpoint:n \l_tmpb_tl
8106   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@_line_width_dim }
8107   \@_qpoint:n { #1 }
8108   \tl_if_empty:NTF \l_@_borders_tikz_tl
8109   {
8110     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8111     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8112     \pgfusepathqstroke
8113   }
8114   {
8115     \use:e { \exp_not:N \draw [ \l_@_borders_tikz_tl ] }
8116     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8117   }
8118 }

```

Here is the set of keys for the command \@_stroke_borders_block:nnn.

```

8119 \keys_define:nn { nicematrix / BlockBorders }
8120 {
8121   borders .clist_set:N = \l_@_borders_clist ,
8122   rounded-corners .dim_set:N = \l_@_rounded_corners_dim ,
8123   rounded-corners .default:n = 4 pt ,
8124   line-width .dim_set:N = \l_@_line_width_dim
8125 }

```

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.

```
8126 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8127 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8128 {
8129   \begin { tikzpicture }
8130   \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because #5 is a list of lists.

```
8131   \clist_map_inline:nn { #1 }
8132   {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8133     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8134     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8135     (
8136     [
8137       xshift = \dim_use:N \l_@@_offset_dim ,
8138       yshift = - \dim_use:N \l_@@_offset_dim
8139     ]
8140     #2 -| #3
8141     )
8142     rectangle
8143     (
8144     [
8145       xshift = - \dim_use:N \l_@@_offset_dim ,
8146       yshift = \dim_use:N \l_@@_offset_dim
8147     ]
8148     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8149     ) ;
8150   }
8151   \end { tikzpicture }
8152 }

8153 \keys_define:nn { nicematrix / SpecialOffset }
8154 { 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.

```
8155 \cs_new_protected:Npn \@@_NullBlock:
8156 { \@@_collect_options:n { \@@_NullBlock_i: } }
8157 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8158 { }
```

27 How to draw the dotted lines transparently

```
8159 \cs_set_protected:Npn \@@_renew_matrix:
8160 {
8161   \RenewDocumentEnvironment { pmatrix } { }
8162   { \pNiceMatrix }
8163   { \endpNiceMatrix }
8164   \RenewDocumentEnvironment { vmatrix } { }
8165   { \vNiceMatrix }
8166   { \endvNiceMatrix }
8167   \RenewDocumentEnvironment { Vmatrix } { }
8168   { \VNiceMatrix }
```

```

8169     { \endVNiceMatrix }
8170 \RenewDocumentEnvironment { bmatrix } { }
8171     { \bNiceMatrix }
8172     { \endbNiceMatrix }
8173 \RenewDocumentEnvironment { Bmatrix } { }
8174     { \BNiceMatrix }
8175     { \endBNiceMatrix }
8176 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8177 \keys_define:nm { nicematrix / Auto }
8178 {
8179     columns-type .tl_set:N = \l_@@_columns_type_tl ,
8180     columns-type .value_required:n = true ,
8181     l .meta:n = { columns-type = l } ,
8182     r .meta:n = { columns-type = r } ,
8183     c .meta:n = { columns-type = c } ,
8184     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8185     delimiters / color .value_required:n = true ,
8186     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8187     delimiters / max-width .default:n = true ,
8188     delimiters .code:n = \keys_set:nm { nicematrix / delimiters } { #1 } ,
8189     delimiters .value_required:n = true ,
8190     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8191     rounded-corners .default:n = 4 pt
8192 }

8193 \NewDocumentCommand \AutoNiceMatrixWithDelims
8194 { m m 0 { } } > { \SplitArgument { 1 } { - } } m 0 { } m ! 0 { } }
8195 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8196 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8197 {

```

The group is for the protection of the keys.

```

8198     \group_begin:
8199     \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8200     \use:e
8201     {
8202         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8203         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8204         [ \exp_not:o \l_tmpa_tl ]
8205     }
8206     \int_if_zero:nT \l_@@_first_row_int
8207     {
8208         \int_if_zero:nT \l_@@_first_col_int { & }
8209         \prg_replicate:nn { #4 - 1 } { & }
8210         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8211     }
8212     \prg_replicate:nn { #3 }
8213     {
8214         \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`).

```

8215         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8216         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8217     }
8218     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8219     {

```

```

8220     \int_if_zero:nT \l_@@_first_col_int { & }
8221     \prg_replicate:nn { #4 - 1 } { & }
8222     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8223   }
8224   \end { NiceArrayWithDelims }
8225   \group_end:
8226 }

8227 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8228 {
8229   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8230   {
8231     \bool_gset_true:N \g_@@_delims_bool
8232     \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8233     \AutoNiceMatrixWithDelims { #2 } { #3 }
8234   }
8235 }

8236 \@@_define_com:nnn p ( )
8237 \@@_define_com:nnn b [ ]
8238 \@@_define_com:nnn v | |
8239 \@@_define_com:nnn V \ | \ |
8240 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8241 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8242 {
8243   \group_begin:
8244   \bool_gset_false:N \g_@@_delims_bool
8245   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8246   \group_end:
8247 }

```

29 The redefinition of the command `\dotfill`

```

8248 \cs_set_eq:NN \@@_old_dotfill \dotfill
8249 \cs_new_protected:Npn \@@_dotfill:
8250 {

```

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}`).

```

8251   \@@_old_dotfill
8252   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8253 }

```

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`.

```

8254 \cs_new_protected:Npn \@@_dotfill_i:
8255 { \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.

```

8256 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8257 {
8258   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

8259     {
8260     \@@_actually_diagbox:nnnnnn
8261     { \int_use:N \c@iRow }
8262     { \int_use:N \c@jCol }
8263     { \int_use:N \c@iRow }
8264     { \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.

```

8265     { \g_@@_row_style_tl \exp_not:n { #1 } }
8266     { \g_@@_row_style_tl \exp_not:n { #2 } }
8267 }

```

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.

```

8268 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8269 {
8270   { \int_use:N \c@iRow }
8271   { \int_use:N \c@jCol }
8272   { \int_use:N \c@iRow }
8273   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8274   { }
8275 }
8276 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8277 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8278 {
8279   \pgfpicture
8280   \pgf@relevantforpicturesizefalse
8281   \pgfrememberpicturepositiononpagetrue
8282   \@@_qpoint:n { row - #1 }
8283   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8284   \@@_qpoint:n { col - #2 }
8285   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8286   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8287   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8288   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8289   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8290   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8291   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8292   {

```

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.

```

8293   \CT@arc@
8294   \pgfsetroundcap
8295   \pgfusepathqstroke
8296 }
8297 \pgfset { inner~sep = 1 pt }
8298 \pgfscope
8299 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8300 \pgfnode { rectangle } { south~west }
8301 {
8302   \begin { minipage } { 20 cm }

```

The `\scan_stop`: avoids an error in math mode when the argument #5 is empty.

```

8303     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8304     \end { minipage }
8305   }
8306   { }
8307   { }
8308   \endpgfscope
8309   \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8310   \pgfnode { rectangle } { north-east }
8311   {
8312     \begin { minipage } { 20 cm }
8313     \raggedleft
8314     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8315     \end { minipage }
8316   }
8317   { }
8318   { }
8319   \endpgfpicture
8320 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8321 \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 `\`.

```

8322 \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`.

```

8323 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8324 {
8325   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8326   \@@_CodeAfter_iv:n
8327 }

```

We catch the argument of the command `\end` (in #1).

```

8328 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8329 {

```

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.

```

8330   \str_if_eq:eeTF \@currenvir { #1 }
8331   { \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`.

```

8332   {
8333     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8334     \@@_CodeAfter_ii:n
8335   }
8336 }

```

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.

```
8337 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8338   {
8339     \pgfpicture
8340     \pgfrememberpicturepositiononpagetrue
8341     \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.

```
8342   \@@_qpoint:n { row - 1 }
8343   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8344   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8345   \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).

```
8346   \bool_if:nTF { #3 }
8347     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8348     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8349   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8350     {
8351       \cs_if_exist:cT
8352         { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8353         {
8354           \pgfpointanchor
8355             { \@@_env: - ##1 - #2 }
8356             { \bool_if:nTF { #3 } { west } { east } }
8357           \dim_set:Nn \l_tmpa_dim
8358             { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8359         }
8360     }
```

Now we can put the delimiter with a node of PGF.

```
8361   \pgfset { inner~sep = \c_zero_dim }
8362   \dim_zero:N \nulldelimiterspace
8363   \pgftransformshift
8364     {
8365       \pgfpoint
8366         { \l_tmpa_dim }
8367         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8368     }
8369   \pgfnode
8370     { rectangle }
8371     { \bool_if:nTF { #3 } { east } { west } }
8372     {
```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```
8373     \nullfont
8374     \c_math_toggle_token
8375     \@@_color:o \l_@@_delimiters_color_tl
8376     \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```

8377     \vcenter
8378     {
8379         \nullfont
8380         \hrule \@height
8381             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8382             \@depth \c_zero_dim
8383             \@width \c_zero_dim
8384     }
8385     \bool_if:nTF { #3 } { \right . } { \right #1 }
8386     \c_math_toggle_token
8387 }
8388 { }
8389 { }
8390 \endpgfpicture
8391 }

```

33 The command `\SubMatrix`

```

8392 \keys_define:nn { nicematrix / sub-matrix }
8393 {
8394     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8395     extra-height .value_required:n = true ,
8396     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8397     left-xshift .value_required:n = true ,
8398     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8399     right-xshift .value_required:n = true ,
8400     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8401     xshift .value_required:n = true ,
8402     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8403     delimiters / color .value_required:n = true ,
8404     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8405     slim .default:n = true ,
8406     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8407     hlines .default:n = all ,
8408     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8409     vlines .default:n = all ,
8410     hvlines .meta:n = { hlines, vlines } ,
8411     hvlines .value_forbidden:n = true
8412 }
8413 \keys_define:nn { nicematrix }
8414 {
8415     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8416     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8417     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8418     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8419 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8420 \keys_define:nn { nicematrix / SubMatrix }
8421 {
8422     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8423     delimiters / color .value_required:n = true ,
8424     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8425     hlines .default:n = all ,
8426     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8427     vlines .default:n = all ,
8428     hvlines .meta:n = { hlines, vlines } ,
8429     hvlines .value_forbidden:n = true ,
8430     name .code:n =

```

```

8431 \tl_if_empty:nTF { #1 }
8432 { \@@_error:n { Invalid-name } }
8433 {
8434   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8435   {
8436     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8437     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8438     {
8439       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8440       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8441     }
8442   }
8443   { \@@_error:n { Invalid-name } }
8444 } ,
8445 name .value_required:n = true ,
8446 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8447 rules .value_required:n = true ,
8448 code .tl_set:N = \l_@@_code_tl ,
8449 code .value_required:n = true ,
8450 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8451 }

8452 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8453 {
8454   \peek_remove_spaces:n
8455   {
8456     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8457     {
8458       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8459       [
8460         delimiters / color = \l_@@_delimiters_color_tl ,
8461         hlines = \l_@@_submatrix_hlines_clist ,
8462         vlines = \l_@@_submatrix_vlines_clist ,
8463         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8464         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8465         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8466         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8467         #5
8468       ]
8469     }
8470     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8471   }
8472 }

8473 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8474 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8475 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8476 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8477 {
8478   \seq_gput_right:Ne \g_@@_submatrix_seq
8479   {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8480     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8481     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8482     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8483     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8484   }
8485 }

```

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`.

```

8486 \hook_gput_code:nnn { begindocument } { . }
8487 {
8488   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8489   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8490   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8491     {
8492       \peek_remove_spaces:n
8493       {
8494         \@@_sub_matrix:nnnnnnn
8495         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8496       }
8497     }
8498 }

```

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).

```

8499 \NewDocumentCommand \@@_compute_i_j:nn
8500 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8501 { \@@_compute_i_j:nnnn #1 #2 }
8502 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8503 {
8504   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8505   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8506   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8507   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8508   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8509     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8510   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8511     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8512   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8513     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8514   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8515     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8516 }
8517 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8518 {
8519   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8520 \@@_compute_i_j:nn { #2 } { #3 }
8521 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8522   { \cs_set_nopar:Npn \arraystretch { 1 } }
8523 \bool_lazy_or:nnTF
8524   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8525   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8526   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8527   {
8528     \str_clear_new:N \l_@@_submatrix_name_str
8529     \keys_set:nn { nicematrix / SubMatrix } { #5 }

```

```

8530 \pgfpicture
8531 \pgfrememberpicturepositiononpagetrue
8532 \pgf@relevantforpicturesizefalse
8533 \pgfset { inner~sep = \c_zero_dim }
8534 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8535 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currfication.

8536 \bool_if:NTF \l_@@_submatrix_slim_bool
8537 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8538 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8539 {
8540 \cs_if_exist:cT
8541 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8542 {
8543 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8544 \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8545 { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8546 }
8547 \cs_if_exist:cT
8548 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8549 {
8550 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8551 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8552 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8553 }
8554 }
8555 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8556 { \@@_error:nn { Impossible~delimiter } { left } }
8557 {
8558 \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8559 { \@@_error:nn { Impossible~delimiter } { right } }
8560 { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8561 }
8562 \endpgfpicture
8563 }
8564 \group_end:
8565 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8566 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8567 {
8568 \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8569 \dim_set:Nn \l_@@_y_initial_dim
8570 {
8571 \fp_to_dim:n
8572 {
8573 \pgf@y
8574 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8575 }
8576 }
8577 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8578 \dim_set:Nn \l_@@_y_final_dim
8579 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8580 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8581 {
8582 \cs_if_exist:cT
8583 { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8584 {
8585 \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8586 \dim_set:Nn \l_@@_y_initial_dim
8587 { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8588 }

```

```

8589     \cs_if_exist:cT
8590     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8591     {
8592         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8593         \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8594         { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8595     }
8596 }
8597 \dim_set:Nn \l_tmpa_dim
8598 {
8599     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8600     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8601 }
8602 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8603     \group_begin:
8604     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8605     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8606     \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`.

```

8607     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8608     {
8609         \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8610         {
8611             \int_compare:nNnT
8612             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8613             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8614                 \@@_qpoint:n { col - ##1 }
8615                 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8616                 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8617                 \pgfusepathqstroke
8618             }
8619         }
8620     }

```

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.

```

8621     \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8622     { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8623     { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8624     {
8625         \bool_lazy_and:nnTF
8626         { \int_compare_p:nNn { ##1 } > \c_zero_int }
8627         {
8628             \int_compare_p:nNn
8629             { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8630         {
8631             \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8632             \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8633             \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8634             \pgfusepathqstroke
8635         }
8636         { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8637     }

```

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.

```

8638 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8639 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8640 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8641 {
8642   \bool_lazy_and:nnTF
8643   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8644   {
8645     \int_compare_p:nNn
8646     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8647   {
8648     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8649   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8650     \dim_set:Nn \l_tmpa_dim
8651     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8652     \str_case:nn { #1 }
8653     {
8654       ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8655       [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8656       \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } } }
8657     }
8658     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8659     \dim_set:Nn \l_tmpb_dim
8660     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8661     \str_case:nn { #2 }
8662     {
8663       ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8664       ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8665       \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8666     }
8667     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8668     \pgfusepathqstroke
8669     \group_end:
8670   }
8671   { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8672 }

```

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).

```

8673 \str_if_empty:NF \l_@@_submatrix_name_str
8674 {
8675   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8676   \l_@@_x_initial_dim \l_@@_y_initial_dim
8677   \l_@@_x_final_dim \l_@@_y_final_dim
8678 }
8679 \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`.

```

8680 \begin { pgfscope }
8681 \pgftransformshift
8682 {
8683   \pgfpoint
8684   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8685   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8686 }
8687 \str_if_empty:NTF \l_@@_submatrix_name_str
8688 { \@@_node_left:nn #1 { } }

```

```

8689     { \l_@@_node_left:nn #1 { \l_@@_env: - \l_@@_submatrix_name_str - left } }
8690 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8691 \pgftransformshift
8692 {
8693   \pgfpoint
8694     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8695     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8696 }
8697 \str_if_empty:NTF \l_@@_submatrix_name_str
8698 { \l_@@_node_right:nnnn #2 { } { #3 } { #4 } }
8699 {
8700   \l_@@_node_right:nnnn #2
8701   { \l_@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8702 }
8703 \cs_set_eq:NN \pgfpointanchor \l_@@_pgfpointanchor:n
8704 \flag_clear_new:N \l_@@_code_flag
8705 \l_@@_code_tl
8706 }

```

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`.

```

8707 \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.

```

8708 \cs_new_protected:Npn \l_@@_pgfpointanchor:n #1
8709 {
8710   \use:e
8711   { \exp_not:N \l_@@_old_pgfpointanchor { \l_@@_pgfpointanchor_i:nn #1 } }
8712 }

```

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 -.

```

8713 \cs_new:Npn \l_@@_pgfpointanchor_i:nn #1 #2
8714 { #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.

```

8715 \tl_const:Nn \c_@@_integers_alist_tl
8716 {
8717   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8718   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8719   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8720   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8721 }
8722 \cs_new:Npn \l_@@_pgfpointanchor_ii:w #1-#2\q_stop
8723 {

```

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`.

```

8724   \tl_if_empty:nTF { #2 }
8725     {
8726       \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8727         {
8728           \flag_raise:N \l_@@_code_flag
8729           \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8730             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8731             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8732         }
8733       { #1 }
8734     }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8735     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8736   }

```

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`).

```

8737 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8738   {
8739     \str_case:nnF { #1 }
8740       {
8741         { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8742         { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8743       }

```

Now the case of a node of the form $i-j$.

```

8744     {
8745       \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8746       - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8747     }
8748   }

```

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`).

```

8749 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8750   {
8751     \pgfnode
8752       { rectangle }
8753       { east }
8754       {
8755         \nullfont
8756         \c_math_toggle_token
8757         \@@_color:o \l_@@_delimiters_color_tl
8758         \left #1
8759         \vcenter
8760           {
8761             \nullfont
8762             \hrule \@height \l_tmpa_dim
8763               \@depth \c_zero_dim
8764               \@width \c_zero_dim
8765           }
8766         \right .
8767         \c_math_toggle_token
8768       }
8769     { #2 }

```

```

8770     { }
8771   }

```

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.

```

8772 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8773 {
8774   \pgfnode
8775     { rectangle }
8776     { west }
8777     {
8778       \nullfont
8779       \c_math_toggle_token
8780       \colorlet { current-color } { . }
8781       \@@_color:o \l_@@_delimiters_color_tl
8782       \left .
8783       \vcenter
8784         {
8785           \nullfont
8786           \hrule \@height \l_tmpa_dim
8787             \@depth \c_zero_dim
8788             \@width \c_zero_dim
8789         }
8790       \right #1
8791       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8792       ~ { \color { current-color } \smash { #4 } }
8793       \c_math_toggle_token
8794     }
8795   { #2 }
8796   { }
8797 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8798 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8799 {
8800   \peek_remove_spaces:n
8801   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8802 }
8803 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8804 {
8805   \peek_remove_spaces:n
8806   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8807 }
8808 \keys_define:nn { nicematrix / Brace }
8809 {
8810   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8811   left-shorten .default:n = true ,
8812   left-shorten .value_forbidden:n = true ,
8813   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8814   right-shorten .default:n = true ,
8815   right-shorten .value_forbidden:n = true ,
8816   shorten .meta:n = { left-shorten , right-shorten } ,
8817   shorten .value_forbidden:n = true ,
8818   yshift .dim_set:N = \l_@@_brace_yshift_dim ,

```

```

8819     yshift .value_required:n = true ,
8820     yshift .initial:n = \c_zero_dim ,
8821     color .tl_set:N = \l_tmpa_tl ,
8822     color .value_required:n = true ,
8823     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8824 }

```

#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.

```

8825 \cs_new_protected:Npn \@@_brace:nnnn #1 #2 #3 #4 #5
8826 {
8827   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8828   \@@_compute_i_j:nn { #1 } { #2 }
8829   \bool_lazy_or:nnTF
8830     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8831     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8832     {
8833       \str_if_eq:eeTF { #5 } { under }
8834       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8835       { \@@_error:nn { Construct-too-large } { \OverBrace } }
8836     }
8837   {
8838     \tl_clear:N \l_tmpa_tl
8839     \keys_set:nn { nicematrix / Brace } { #4 }
8840     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8841     \pgfpicture
8842     \pgfrememberpicturepositiononpagetrue
8843     \pgf@relevantforpicturesizefalse
8844     \bool_if:NT \l_@@_brace_left_shorten_bool
8845     {
8846       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8847       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8848       {
8849         \cs_if_exist:cT
8850           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8851           {
8852             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8853           }
8854         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8855           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8856       }
8857     }
8858   }
8859   \bool_lazy_or:nnT
8860     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8861     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8862     {
8863       \@@_qpoint:n { col - \l_@@_first_j_tl }
8864       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8865     }
8866   \bool_if:NT \l_@@_brace_right_shorten_bool
8867   {
8868     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8869     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8870     {
8871       \cs_if_exist:cT
8872         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8873         {
8874           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8875           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8876             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }

```

```

8877     }
8878   }
8879 }
8880 \bool_lazy_or:nnT
8881 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8882 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8883 {
8884   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8885   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8886 }
8887 \pgfset { inner~sep = \c_zero_dim }
8888 \str_if_eq:eeTF { #5 } { under }
8889 { \@@_underbrace_i:n { #3 } }
8890 { \@@_overbrace_i:n { #3 } }
8891 \endpgfpicture
8892 }
8893 \group_end:
8894 }

```

The argument is the text to put above the brace.

```

8895 \cs_new_protected:Npn \@@_overbrace_i:n #1
8896 {
8897   \@@_qpoint:n { row - \l_@@_first_i_tl }
8898   \pgftransformshift
8899   {
8900     \pgfpoint
8901     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8902     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8903   }
8904   \pgfnode
8905   { rectangle }
8906   { south }
8907   {
8908     \vtop
8909     {
8910       \group_begin:
8911       \everycr { }
8912       \halign
8913       {
8914         \hfil ## \hfil \crcr
8915         \bool_if:NTF \l_@@_tabular_bool
8916         { \begin { tabular } { c } #1 \end { tabular } }
8917         { $ \begin { array } { c } #1 \end { array } $ }
8918         \cr
8919         \c_math_toggle_token
8920         \overbrace
8921         {
8922           \hbox_to_wd:nn
8923           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8924           { }
8925         }
8926         \c_math_toggle_token
8927         \cr
8928       }
8929       \group_end:
8930     }
8931   }
8932   { }
8933   { }
8934 }

```

The argument is the text to put under the brace.

```

8935 \cs_new_protected:Npn \@@_underbrace_i:n #1

```

```

8936 {
8937   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8938   \pgftransformshift
8939     {
8940       \pgfpoint
8941         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8942         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8943     }
8944   \pgfnode
8945     { rectangle }
8946     { north }
8947     {
8948       \group_begin:
8949       \everycr { }
8950       \vbox
8951         {
8952           \halign
8953             {
8954               \hfil ## \hfil \crcr
8955               \c_math_toggle_token
8956               \underbrace
8957                 {
8958                   \hbox_to_wd:nn
8959                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8960                     { }
8961                 }
8962               \c_math_toggle_token
8963               \cr
8964               \bool_if:NTF \l_@@_tabular_bool
8965                 { \begin { tabular } { c } #1 \end { tabular } }
8966                 { $ \begin { array } { c } #1 \end { array } $ }
8967               \cr
8968             }
8969         }
8970       \group_end:
8971     }
8972   { }
8973   { }
8974 }

```

35 The command TikzEveryCell

```

8975 \bool_new:N \l_@@_not_empty_bool
8976 \bool_new:N \l_@@_empty_bool
8977
8978 \keys_define:nn { nicematrix / TikzEveryCell }
8979 {
8980   not-empty .code:n =
8981     \bool_lazy_or:nnTF
8982       \l_@@_in_code_after_bool
8983       \g_@@_recreate_cell_nodes_bool
8984       { \bool_set_true:N \l_@@_not_empty_bool }
8985       { \@@_error:n { detection-of-empty-cells } } ,
8986   not-empty .value_forbidden:n = true ,
8987   empty .code:n =
8988     \bool_lazy_or:nnTF
8989       \l_@@_in_code_after_bool
8990       \g_@@_recreate_cell_nodes_bool
8991       { \bool_set_true:N \l_@@_empty_bool }

```

```

8992     { \@@_error:n { detection~of~empty~cells } } ,
8993     empty .value_forbidden:n = true ,
8994     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8995 }

```

```

8996
8997
8998 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8999 {
9000   \IfPackageLoadedTF { tikz }
9001   {
9002     \group_begin:
9003     \keys_set:nm { 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.

```

9004     \tl_set:Nn \l_tmpa_tl { { #2 } }
9005     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9006     { \@@_for_a_block:nnnnn ##1 }
9007     \@@_all_the_cells:
9008     \group_end:
9009   }
9010   { \@@_error:n { TikzEveryCell~without~tikz } }
9011 }
9012
9013 \tl_new:N \@@_i_tl
9014 \tl_new:N \@@_j_tl
9015
9016
9017 \cs_new_protected:Nn \@@_all_the_cells:
9018 {
9019   \int_step_variable:nNn \c@iRow \@@_i_tl
9020   {
9021     \int_step_variable:nNn \c@jCol \@@_j_tl
9022     {
9023       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9024       {
9025         \clist_if_in:Nef \l_@@_corners_cells_clist
9026         { \@@_i_tl - \@@_j_tl }
9027         {
9028           \bool_set_false:N \l_tmpa_bool
9029           \cs_if_exist:cTF
9030           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9031           {
9032             \bool_if:Nf \l_@@_empty_bool
9033             { \bool_set_true:N \l_tmpa_bool }
9034           }
9035           {
9036             \bool_if:Nf \l_@@_not_empty_bool
9037             { \bool_set_true:N \l_tmpa_bool }
9038           }
9039           \bool_if:NT \l_tmpa_bool
9040           {
9041             \@@_block_tikz:nnnnn
9042             \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9043           }
9044         }
9045       }
9046     }
9047   }
9048 }
9049
9050 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9051 {
9052   \bool_if:Nf \l_@@_empty_bool

```

```

9053     {
9054         \@@_block_tikz:onnnn
9055         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9056     }
9057     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9058 }
9059
9060 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9061 {
9062     \int_step_inline:nnn { #1 } { #3 }
9063     {
9064         \int_step_inline:nnn { #2 } { #4 }
9065         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9066     }
9067 }

```

36 The command \ShowCellNames

```

9068 \NewDocumentCommand \@@_ShowCellNames { }
9069 {
9070     \bool_if:NT \l_@@_in_code_after_bool
9071     {
9072         \pgfpicture
9073         \pgfrememberpicturepositiononpagetrue
9074         \pgf@relevantforpicturesizefalse
9075         \pgfpathrectanglecorners
9076         { \@@_qpoint:n { 1 } }
9077         {
9078             \@@_qpoint:n
9079             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9080         }
9081         \pgfsetfillopacity { 0.75 }
9082         \pgfsetfillcolor { white }
9083         \pgfusepathqfill
9084         \endpgfpicture
9085     }
9086     \dim_gzero_new:N \g_@@_tmpc_dim
9087     \dim_gzero_new:N \g_@@_tmpd_dim
9088     \dim_gzero_new:N \g_@@_tmpe_dim
9089     \int_step_inline:nn \c@iRow
9090     {
9091         \bool_if:NTF \l_@@_in_code_after_bool
9092         {
9093             \pgfpicture
9094             \pgfrememberpicturepositiononpagetrue
9095             \pgf@relevantforpicturesizefalse
9096         }
9097         { \begin { pgfpicture } }
9098         \@@_qpoint:n { row - ##1 }
9099         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9100         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9101         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9102         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9103         \bool_if:NTF \l_@@_in_code_after_bool
9104         { \endpgfpicture }
9105         { \end { pgfpicture } }
9106         \int_step_inline:nn \c@jCol
9107         {
9108             \hbox_set:Nn \l_tmpa_box
9109             {
9110                 \normalfont \Large \sffamily \bfseries
9111                 \bool_if:NTF \l_@@_in_code_after_bool

```

```

9112         { \color { red } }
9113         { \color { red ! 50 } }
9114     ##1 - ####1
9115     }
9116 \bool_if:NTF \l_@@_in_code_after_bool
9117     {
9118     \pgfpicture
9119     \pgfrememberpicturepositiononpagetrue
9120     \pgf@relevantforpicturesizefalse
9121     }
9122     { \begin { pgfpicture } }
9123     \@@_qpoint:n { col - ####1 }
9124     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9125     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9126     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9127     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9128     \bool_if:NTF \l_@@_in_code_after_bool
9129     { \endpgfpicture }
9130     { \end { pgfpicture } }
9131     \fp_set:Nn \l_tmpa_fp
9132     {
9133     \fp_min:nn
9134     {
9135     \fp_min:nn
9136     { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9137     { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9138     }
9139     { 1.0 }
9140     }
9141     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9142     \pgfpicture
9143     \pgfrememberpicturepositiononpagetrue
9144     \pgf@relevantforpicturesizefalse
9145     \pgftransformshift
9146     {
9147     \pgfpoint
9148     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9149     { \dim_use:N \g_tmpa_dim }
9150     }
9151     \pgfnode
9152     { rectangle }
9153     { center }
9154     { \box_use:N \l_tmpa_box }
9155     { }
9156     { }
9157     \endpgfpicture
9158     }
9159     }
9160 }

```

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.

```

9161 \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.

```

9162 \bool_new:N \g_@@_footnote_bool
9163 \msg_new:nmmn { nicematrix } { Unknown~key~for~package }
9164 {
9165   The~key~'\l_keys_key_str'~is~unknown. \\
9166   That~key~will~be~ignored. \\
9167   For~a~list~of~the~available~keys,~type~H~<return>.
9168 }
9169 {
9170   The~available~keys~are~(in~alphabetic~order):~
9171   footnote,~
9172   footnotehyper,~
9173   messages~for~Overleaf,~
9174   renew~dots,~and~
9175   renew~matrix.
9176 }
9177 \keys_define:nn { nicematrix / Package }
9178 {
9179   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9180   renew~dots .value_forbidden:n = true ,
9181   renew~matrix .code:n = \@@_renew_matrix: ,
9182   renew~matrix .value_forbidden:n = true ,
9183   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9184   footnote .bool_set:N = \g_@@_footnote_bool ,
9185   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.

```

9186   no~test~for~array .code:n = \prg_do_nothing: ,
9187   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9188 }
9189 \ProcessKeysOptions { nicematrix / Package }

9190 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9191 {
9192   You~can't~use~the~option~'footnote'~because~the~package~
9193   footnotehyper~has~already~been~loaded.~
9194   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9195   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9196   of~the~package~footnotehyper.\\
9197   The~package~footnote~won't~be~loaded.
9198 }
9199 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9200 {
9201   You~can't~use~the~option~'footnotehyper'~because~the~package~
9202   footnote~has~already~been~loaded.~
9203   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9204   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9205   of~the~package~footnote.\\
9206   The~package~footnotehyper~won't~be~loaded.
9207 }

9208 \bool_if:NT \g_@@_footnote_bool
9209 {
9210   \IfClassLoadedTF { beamer }
9211   { \bool_set_false:N \g_@@_footnote_bool }
9212   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9213     \IfPackageLoadedTF { footnotehyper }
9214     { \@@_error:n { footnote~with~footnotehyper~package } }
9215     { \usepackage { footnote } }
9216   }
9217 }
9218 \bool_if:NT \g_@@_footnotehyper_bool
9219 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9220   \IfClassLoadedTF { beamer }
9221   { \bool_set_false:N \g_@@_footnote_bool }
9222   {
9223     \IfPackageLoadedTF { footnote }
9224     { \@@_error:n { footnotehyper~with~footnote~package } }
9225     { \usepackage { footnotehyper } }
9226   }
9227   \bool_set_true:N \g_@@_footnote_bool
9228 }

```

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.

```

9229 \bool_new:N \l_@@_underscore_loaded_bool
9230 \IfPackageLoadedT { underscore }
9231 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9232 \hook_gput_code:nnn { begindocument } { . }
9233 {
9234   \bool_if:NF \l_@@_underscore_loaded_bool
9235   {
9236     \IfPackageLoadedT { underscore }
9237     { \@@_error:n { underscore~after~nicematrix } }
9238   }
9239 }

```

39 Error messages of the package

```

9240 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9241 { \str_const:Nn \c_@@_available_keys_str { } }
9242 {
9243   \str_const:Nn \c_@@_available_keys_str
9244   { For~a~list~of~the~available~keys,~type~H~<return>. }
9245 }
9246 \seq_new:N \g_@@_types_of_matrix_seq
9247 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9248 {
9249   NiceMatrix ,
9250   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9251 }
9252 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq

```

```
9253 { \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`.

```
9254 \cs_new_protected:Npn \@@_error_too_much_cols:
9255 {
9256   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9257   { \@@_fatal:n { too-much-cols-for-array } }
9258   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9259   { \@@_fatal:n { too-much-cols-for-matrix } }
9260   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9261   { \@@_fatal:n { too-much-cols-for-matrix } }
9262   \bool_if:NF \l_@@_last_col_without_value_bool
9263   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9264 }
```

The following command must *not* be protected since it's used in an error message.

```
9265 \cs_new:Npn \@@_message_hdotsfor:
9266 {
9267   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9268   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9269 }
9270 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9271 {
9272   Incompatible~options.\\
9273   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9274   The~output~will~not~be~reliable.
9275 }
9276 \@@_msg_new:nn { key~color~inside }
9277 {
9278   Key~deprecated.\\
9279   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9280   and~have~been~deprecated.\\
9281   You~won't~have~similar~message~till~the~end~of~the~document.
9282 }
9283 \@@_msg_new:nn { negative~weight }
9284 {
9285   Negative~weight.\\
9286   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9287   the~value~'\int_use:N \l_@@_weight_int'.\\
9288   The~absolute~value~will~be~used.
9289 }
9290 \@@_msg_new:nn { last~col~not~used }
9291 {
9292   Column~not~used.\\
9293   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9294   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9295 }
9296 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9297 {
9298   Too~much~columns.\\
9299   In~the~row~\int_eval:n { \c@iRow },~
9300   you~try~to~use~more~columns~
9301   than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9302   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9303   (plus~the~exterior~columns).~This~error~is~fatal.
9304 }
9305 \@@_msg_new:nn { too~much~cols~for~matrix }
9306 {
```

```

9307 Too-much-columns.\\
9308 In~the~row~\int_eval:n { \c@iRow },~
9309 you~try~to~use~more~columns~than~allowed~by~your~
9310 \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9311 number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9312 columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9313 Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9314 \token_to_str:N \setcounter\ to~change~that~value).~
9315 This~error~is~fatal.
9316 }

9317 \@@_msg_new:nn { too-much-cols-for-array }
9318 {
9319 Too-much-columns.\\
9320 In~the~row~\int_eval:n { \c@iRow },~
9321 ~you~try~to~use~more~columns~than~allowed~by~your~
9322 \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9323 \int_use:N \g_@@_static_num_of_col_int\
9324 ~ (plus~the~potential~exterior~ones).~
9325 This~error~is~fatal.
9326 }

9327 \@@_msg_new:nn { columns-not-used }
9328 {
9329 Columns~not~used.\\
9330 The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9331 \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9332 The~columns~you~did~not~used~won't~be~created.\\
9333 You~won't~have~similar~error~message~till~the~end~of~the~document.
9334 }

9335 \@@_msg_new:nn { empty-preamble }
9336 {
9337 Empty~preamble.\\
9338 The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9339 This~error~is~fatal.
9340 }

9341 \@@_msg_new:nn { in~first~col }
9342 {
9343 Erroneous~use.\\
9344 You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9345 That~command~will~be~ignored.
9346 }

9347 \@@_msg_new:nn { in~last~col }
9348 {
9349 Erroneous~use.\\
9350 You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9351 That~command~will~be~ignored.
9352 }

9353 \@@_msg_new:nn { in~first~row }
9354 {
9355 Erroneous~use.\\
9356 You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9357 That~command~will~be~ignored.
9358 }

9359 \@@_msg_new:nn { in~last~row }
9360 {
9361 You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9362 That~command~will~be~ignored.
9363 }

9364 \@@_msg_new:nn { caption-outside-float }
9365 {
9366 Key~caption~forbidden.\\

```

```

9367     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9368     environment.~This~key~will~be~ignored.
9369   }
9370   \@@_msg_new:nn { short-caption-without-caption }
9371   {
9372     You~should~not~use~the~key~'short-caption'~without~'caption'.~
9373     However,~your~'short-caption'~will~be~used~as~'caption'.
9374   }
9375   \@@_msg_new:nn { double-closing-delimiter }
9376   {
9377     Double~delimiter.\\
9378     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9379     delimiter.~This~delimiter~will~be~ignored.
9380   }
9381   \@@_msg_new:nn { delimiter-after-opening }
9382   {
9383     Double~delimiter.\\
9384     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9385     delimiter.~That~delimiter~will~be~ignored.
9386   }
9387   \@@_msg_new:nn { bad-option-for-line-style }
9388   {
9389     Bad~line~style.\\
9390     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9391     is~'standard'.~That~key~will~be~ignored.
9392   }
9393   \@@_msg_new:nn { Identical-notes-in-caption }
9394   {
9395     Identical~tabular~notes.\\
9396     You~can't~put~several~notes~with~the~same~content~in~
9397     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9398     If~you~go~on,~the~output~will~probably~be~erroneous.
9399   }
9400   \@@_msg_new:nn { tabularnote-below-the-tabular }
9401   {
9402     \token_to_str:N \tabularnote\ forbidden\\
9403     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9404     of~your~tabular~because~the~caption~will~be~composed~below~
9405     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9406     key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
9407     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9408     no~similar~error~will~raised~in~this~document.
9409   }
9410   \@@_msg_new:nn { Unknown-key-for-rules }
9411   {
9412     Unknown~key.\\
9413     There~is~only~two~keys~available~here:~width~and~color.\\
9414     Your~key~'\l_keys_key_str'~will~be~ignored.
9415   }
9416   \@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
9417   {
9418     Unknown~key.\\
9419     There~is~only~two~keys~available~here:~
9420     'empty'~and~'not-empty'.\\
9421     Your~key~'\l_keys_key_str'~will~be~ignored.
9422   }
9423   \@@_msg_new:nn { Unknown-key-for-rotate }
9424   {
9425     Unknown~key.\\
9426     The~only~key~available~here~is~'c'.\\

```

```

9427     Your-key-\l_keys_key_str'-will-be-ignored.
9428 }
9429 \@@_msg_new:nnn { Unknown-key-for-custom-line }
9430 {
9431     Unknown-key.\
9432     The-key-\l_keys_key_str'~is-unknown-in-a-'custom-line'.~
9433     It~you-go-on,~you-will-probably-have-other-errors. \
9434     \c_@@_available_keys_str
9435 }
9436 {
9437     The-available-keys-are-(in-alphabetic-order):~
9438     ccommand,~
9439     color,~
9440     command,~
9441     dotted,~
9442     letter,~
9443     multiplicity,~
9444     sep-color,~
9445     tikz,~and~total-width.
9446 }
9447 \@@_msg_new:nnn { Unknown-key-for-xdots }
9448 {
9449     Unknown-key.\
9450     The-key-\l_keys_key_str'~is~unknown-for~a~command~for~drawing~dotted~rules.\
9451     \c_@@_available_keys_str
9452 }
9453 {
9454     The-available-keys-are-(in-alphabetic-order):~
9455     'color',~
9456     'horizontal-labels',~
9457     'inter',~
9458     'line-style',~
9459     'radius',~
9460     'shorten',~
9461     'shorten-end'~and~'shorten-start'.
9462 }
9463 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9464 {
9465     Unknown-key.\
9466     As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
9467     (and-you-try-to-use-\l_keys_key_str')\
9468     That-key-will-be-ignored.
9469 }
9470 \@@_msg_new:nn { label-without-caption }
9471 {
9472     You-can't-use-the-key~'label'~in-your~{NiceTabular}'~because~
9473     you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9474 }
9475 \@@_msg_new:nn { W-warning }
9476 {
9477     Line-\msg_line_number:.~The-cell-is-too-wide-for-your-column~'W'~
9478     (row~\int_use:N \c@iRow).
9479 }
9480 \@@_msg_new:nn { Construct-too-large }
9481 {
9482     Construct-too-large.\
9483     Your-command~\token_to_str:N #1
9484     can't-be-drawn~because~your~matrix~is~too~small.\
9485     That~command~will~be~ignored.
9486 }
9487 \@@_msg_new:nn { underscore-after-nicematrix }

```

```

9488 {
9489   Problem-with~'underscore'.\\
9490   The-package~'underscore'~should-be~loaded-before~'nicematrix'.~
9491   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9492   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9493 }
9494 \@@_msg_new:nn { ampersand-in~light-syntax }
9495 {
9496   Ampersand~forbidden.\\
9497   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns~because~
9498   ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9499 }
9500 \@@_msg_new:nn { double-backslash-in-light-syntax }
9501 {
9502   Double~backslash~forbidden.\\
9503   You-can't-use~\token_to_str:N
9504   \\~to-separate-rows~because~the~key~'light-syntax'~
9505   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9506   (set~by~the~key~'end-of-row').~This-error-is-fatal.
9507 }
9508 \@@_msg_new:nn { hlines-with-color }
9509 {
9510   Incompatible-keys.\\
9511   You-can't-use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9512   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9513   However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9514   Your-key~will~be~discarded.
9515 }
9516 \@@_msg_new:nn { bad-value-for-baseline }
9517 {
9518   Bad-value-for-baseline.\\
9519   The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9520   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9521   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9522   the~form~'line-i'.\\
9523   A~value~of~1~will~be~used.
9524 }
9525 \@@_msg_new:nn { detection-of~empty~cells }
9526 {
9527   Problem-with~'not-empty'\\
9528   For~technical~reasons,~you~must~activate~
9529   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9530   in~order~to~use~the~key~'\l_keys_key_str'.\\
9531   That~key~will~be~ignored.
9532 }
9533 \@@_msg_new:nn { siunitx-not-loaded }
9534 {
9535   siunitx-not-loaded\\
9536   You-can't-use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9537   That~error~is~fatal.
9538 }
9539 \@@_msg_new:nn { Invalid-name }
9540 {
9541   Invalid-name.\\
9542   You-can't-give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9543   \SubMatrix\ of~your~\@@_full_name_env:.\\
9544   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9545   This~key~will~be~ignored.
9546 }
9547 \@@_msg_new:nn { Wrong~line~in~SubMatrix }

```

```

9548 {
9549   Wrong~line.\\
9550   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9551   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9552   number~is~not~valid.~It~will~be~ignored.
9553 }

9554 \@@_msg_new:nn { Impossible-delimiter }
9555 {
9556   Impossible-delimiter.\\
9557   It's~impossible~to~draw~the~#1~delimiter~of~your~
9558   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9559   in~that~column.
9560   \bool_if:NT \l_@@_submatrix_slim_bool
9561   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9562   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9563 }

9564 \@@_msg_new:nnn { width-without-X-columns }
9565 {
9566   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9567   That~key~will~be~ignored.
9568 }
9569 {
9570   This~message~is~the~message~'width~without~X~columns'~
9571   of~the~module~'nicematrix'.~
9572   The~experimented~users~can~disable~that~message~with~
9573   \token_to_str:N \msg_redirect_name:nnn.\\
9574 }
9575

9576 \@@_msg_new:nn { key-multiplicity-with-dotted }
9577 {
9578   Incompatible~keys. \\
9579   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9580   in~a~'custom-line'.~They~are~incompatible. \\
9581   The~key~'multiplicity'~will~be~discarded.
9582 }

9583 \@@_msg_new:nn { empty-environment }
9584 {
9585   Empty~environment.\\
9586   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9587 }

9588 \@@_msg_new:nn { No~letter~and~no~command }
9589 {
9590   Erroneous~use.\\
9591   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9592   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9593   '~command'~(to~draw~horizontal~rules).\\
9594   However,~you~can~go~on.
9595 }

9596 \@@_msg_new:nn { Forbidden-letter }
9597 {
9598   Forbidden~letter.\\
9599   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9600   It~will~be~ignored.
9601 }

9602 \@@_msg_new:nn { Several~letters }
9603 {
9604   Wrong~name.\\
9605   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9606   have~used~'\l_@@_letter_str').\\
9607   It~will~be~ignored.
9608 }

```

```

9609 \@@_msg_new:nn { Delimiter-with-small }
9610 {
9611   Delimiter~forbidden.\\
9612   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9613   because~the~key~'small'~is~in~force.\\
9614   This~error~is~fatal.
9615 }

9616 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9617 {
9618   Unknown~cell.\\
9619   Your~command~\token_to_str:N\line\{#1}\{#2}\~in~
9620   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9621   can't~be~executed~because~a~cell~doesn't~exist.\\
9622   This~command~\token_to_str:N \line\ will~be~ignored.
9623 }

9624 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9625 {
9626   Duplicate~name.\\
9627   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9628   in~this~\@@_full_name_env:.\\
9629   This~key~will~be~ignored.\\
9630   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9631   { For~a~list~of~the~names~already~used,~type-H<return>. }
9632 }
9633 {
9634   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9635   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9636 }

9637 \@@_msg_new:nn { r-or-l-with-preamble }
9638 {
9639   Erroneous~use.\\
9640   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~.~
9641   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9642   your~\@@_full_name_env:.\\
9643   This~key~will~be~ignored.
9644 }

9645 \@@_msg_new:nn { Hdotsfor-in-col-0 }
9646 {
9647   Erroneous~use.\\
9648   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9649   the~array.~This~error~is~fatal.
9650 }

9651 \@@_msg_new:nn { bad-corner }
9652 {
9653   Bad~corner.\\
9654   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9655   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9656   This~specification~of~corner~will~be~ignored.
9657 }

9658 \@@_msg_new:nn { bad-border }
9659 {
9660   Bad~border.\\
9661   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9662   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9663   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9664   also~use~the~key~'tikz'
9665   \IfPackageLoadedF { tikz
9666     {~if~you~load~the~LaTeX~package~'tikz'})\\
9667   This~specification~of~border~will~be~ignored.
9668 }

9669 \@@_msg_new:nn { TikzEveryCell-without-tikz }

```

```

9670 {
9671   TikZ~not~loaded.\\
9672   You~can't~use~\token_to_str:N \TikzEveryCell\
9673   because~you~have~not~loaded~tikz.~
9674   This~command~will~be~ignored.
9675 }

9676 \@@_msg_new:nn { tikz~key~without~tikz }
9677 {
9678   TikZ~not~loaded.\\
9679   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9680   \Block'~because~you~have~not~loaded~tikz.~
9681   This~key~will~be~ignored.
9682 }

9683 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9684 {
9685   Erroneous~use.\\
9686   In~the~\@@_full_name_env:,~you~must~use~the~key~
9687   'last~col'~without~value.\\
9688   However,~you~can~go~on~for~this~time~
9689   (the~value~'\l_keys_value_tl'~will~be~ignored).
9690 }

9691 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9692 {
9693   Erroneous~use.\\
9694   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9695   'last~col'~without~value.\\
9696   However,~you~can~go~on~for~this~time~
9697   (the~value~'\l_keys_value_tl'~will~be~ignored).
9698 }

9699 \@@_msg_new:nn { Block~too~large~1 }
9700 {
9701   Block~too~large.\\
9702   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
9703   too~small~for~that~block. \\
9704   This~block~and~maybe~others~will~be~ignored.
9705 }

9706 \@@_msg_new:nn { Block~too~large~2 }
9707 {
9708   Block~too~large.\\
9709   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9710   \g_@@_static_num_of_col_int\
9711   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9712   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
9713   (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9714   This~block~and~maybe~others~will~be~ignored.
9715 }

9716 \@@_msg_new:nn { unknown~column~type }
9717 {
9718   Bad~column~type.\\
9719   The~column~type~'#1'~in~your~\@@_full_name_env:\
9720   is~unknown. \\
9721   This~error~is~fatal.
9722 }

9723 \@@_msg_new:nn { unknown~column~type~S }
9724 {
9725   Bad~column~type.\\
9726   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9727   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9728   load~that~package. \\
9729   This~error~is~fatal.
9730 }

```

```

9731 \@@_msg_new:nn { tabularnote~forbidden }
9732 {
9733   Forbidden~command.\
9734   You~can't~use~the~command~\token_to_str:N\tabularnote\
9735   ~here.~This~command~is~available~only~in~
9736   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9737   the~argument~of~a~command~\token_to_str:N \caption\ included~
9738   in~an~environment~{table}. \
9739   This~command~will~be~ignored.
9740 }

9741 \@@_msg_new:nn { borders~forbidden }
9742 {
9743   Forbidden~key.\
9744   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9745   because~the~option~'rounded~corners'~
9746   is~in~force~with~a~non~zero~value.\
9747   This~key~will~be~ignored.
9748 }

9749 \@@_msg_new:nn { bottomrule~without~booktabs }
9750 {
9751   booktabs~not~loaded.\
9752   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9753   loaded~'booktabs'.\
9754   This~key~will~be~ignored.
9755 }

9756 \@@_msg_new:nn { enumitem~not~loaded }
9757 {
9758   enumitem~not~loaded.\
9759   You~can't~use~the~command~\token_to_str:N\tabularnote\
9760   ~because~you~haven't~loaded~'enumitem'.\
9761   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9762   ignored~in~the~document.
9763 }

9764 \@@_msg_new:nn { tikz~without~tikz }
9765 {
9766   Tikz~not~loaded.\
9767   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9768   loaded.~If~you~go~on,~that~key~will~be~ignored.
9769 }

9770 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9771 {
9772   Tikz~not~loaded.\
9773   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9774   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9775   You~can~go~on~but~you~will~have~another~error~if~you~actually~
9776   use~that~custom~line.
9777 }

9778 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9779 {
9780   Tikz~not~loaded.\
9781   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9782   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9783   That~key~will~be~ignored.
9784 }

9785 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9786 {
9787   Erroneous~use.\
9788   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9789   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9790   The~key~'color'~will~be~discarded.
9791 }

```

```

9792 \@@_msg_new:nn { Wrong-last-row }
9793 {
9794   Wrong-number.\
9795   You-have-used-'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9796   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9797   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9798   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9799   without~value~(more~compilations~might~be~necessary).
9800 }
9801 \@@_msg_new:nn { Yet-in-env }
9802 {
9803   Nested-environments.\
9804   Environments-of~nicematrix~can't~be~nested.\
9805   This~error~is~fatal.
9806 }
9807 \@@_msg_new:nn { Outside-math-mode }
9808 {
9809   Outside-math-mode.\
9810   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9811   (and~not~in~\token_to_str:N \vcenter).\
9812   This~error~is~fatal.
9813 }
9814 \@@_msg_new:nn { One-letter~allowed }
9815 {
9816   Bad-name.\
9817   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\
9818   It~will~be~ignored.
9819 }
9820 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9821 {
9822   Environment~{TabularNote}~forbidden.\
9823   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9824   but~*before*~the~\token_to_str:N \CodeAfter.\
9825   This~environment~{TabularNote}~will~be~ignored.
9826 }
9827 \@@_msg_new:nn { varwidth~not~loaded }
9828 {
9829   varwidth~not~loaded.\
9830   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9831   loaded.\
9832   Your~column~will~behave~like~'p'.
9833 }
9834 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9835 {
9836   Unknow~key.\
9837   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
9838   \c_@@_available_keys_str
9839 }
9840 {
9841   The~available~keys~are~(in~alphabetic~order):~
9842   color,~
9843   dotted,~
9844   multiplicity,~
9845   sep-color,~
9846   tikz,~and~total~width.
9847 }
9848
9849 \@@_msg_new:nnn { Unknown~key~for~Block }
9850 {
9851   Unknown~key.\
9852   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N

```

```

9853 \Block.\ It-will-be-ignored. \
9854 \c_@@_available_keys_str
9855 }
9856 {
9857 The-available-keys-are-(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
9858 b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9859 opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9860 and~vlines.
9861 }
9862 \@@_msg_new:nnn { Unknown-key-for-Brace }
9863 {
9864 Unknown-key.\
9865 The-key~'\l_keys_key_str'~is-unknown-for-the~commands~\token_to_str:N
9866 \UnderBrace\ and~\token_to_str:N \OverBrace.\
9867 It-will-be-ignored. \
9868 \c_@@_available_keys_str
9869 }
9870 {
9871 The-available-keys-are-(in-alphabetic-order):~color,~left-shorten,~
9872 right-shorten,~shorten-(which-fixes-both-left-shorten-and~
9873 right-shorten)~and-yshift.
9874 }
9875 \@@_msg_new:nnn { Unknown-key-for-CodeAfter }
9876 {
9877 Unknown-key.\
9878 The-key~'\l_keys_key_str'~is-unknown.\
9879 It-will-be-ignored. \
9880 \c_@@_available_keys_str
9881 }
9882 {
9883 The-available-keys-are-(in-alphabetic-order):~
9884 delimiters/color,~
9885 rules-(with-the-subkeys-'color'~and-'width'),~
9886 sub-matrix-(several-subkeys)~
9887 and~xdots-(several-subkeys).~
9888 The-latter-is-for-the-command~\token_to_str:N \line.
9889 }
9890 \@@_msg_new:nnn { Unknown-key-for-CodeBefore }
9891 {
9892 Unknown-key.\
9893 The-key~'\l_keys_key_str'~is-unknown.\
9894 It-will-be-ignored. \
9895 \c_@@_available_keys_str
9896 }
9897 {
9898 The-available-keys-are-(in-alphabetic-order):~
9899 create-cell-nodes,~
9900 delimiters/color~and~
9901 sub-matrix-(several-subkeys).
9902 }
9903 \@@_msg_new:nnn { Unknown-key-for-SubMatrix }
9904 {
9905 Unknown-key.\
9906 The-key~'\l_keys_key_str'~is-unknown.\
9907 That-key-will-be-ignored. \
9908 \c_@@_available_keys_str
9909 }
9910 {
9911 The-available-keys-are-(in-alphabetic-order):~
9912 'delimiters/color',~
9913 'extra-height',~
9914 'hlines',~

```

```

9915 'hvlines',~
9916 'left-xshift',~
9917 'name',~
9918 'right-xshift',~
9919 'rules'~(with-the-subkeys~'color'~and~'width'),~
9920 'slim',~
9921 'vlines'~and~'xshift'~(which-sets-both~'left-xshift'~
9922 and~'right-xshift').\\
9923 }
9924 \@@_msg_new:nnn { Unknown~key~for~notes }
9925 {
9926   Unknown~key.\\
9927   The~key~'\l_keys_key_str'~is~unknown.\\
9928   That~key~will~be~ignored. \\
9929   \c_@@_available_keys_str
9930 }
9931 {
9932   The~available~keys~are~(in~alphabetic~order):~
9933   bottomrule,~
9934   code-after,~
9935   code-before,~
9936   detect-duplicates,~
9937   enumitem-keys,~
9938   enumitem-keys-para,~
9939   para,~
9940   label-in-list,~
9941   label-in-tabular~and~
9942   style.
9943 }
9944 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9945 {
9946   Unknown~key.\\
9947   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9948   \token_to_str:N \RowStyle. \\
9949   That~key~will~be~ignored. \\
9950   \c_@@_available_keys_str
9951 }
9952 {
9953   The~available~keys~are~(in~alphabetic~order):~
9954   bold,~
9955   cell-space-top-limit,~
9956   cell-space-bottom-limit,~
9957   cell-space-limits,~
9958   color,~
9959   fill~(alias:~rowcolor),~
9960   nb-rows,
9961   opacity~and~
9962   rounded-corners.
9963 }
9964 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9965 {
9966   Unknown~key.\\
9967   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9968   \token_to_str:N \NiceMatrixOptions. \\
9969   That~key~will~be~ignored. \\
9970   \c_@@_available_keys_str
9971 }
9972 {
9973   The~available~keys~are~(in~alphabetic~order):~
9974   &-in-blocks,~
9975   allow-duplicate-names,~
9976   ampersand-in-blocks,~
9977   caption-above,~

```

```

9978   cell-space-bottom-limit,~
9979   cell-space-limits,~
9980   cell-space-top-limit,~
9981   code-for-first-col,~
9982   code-for-first-row,~
9983   code-for-last-col,~
9984   code-for-last-row,~
9985   corners,~
9986   custom-key,~
9987   create-extra-nodes,~
9988   create-medium-nodes,~
9989   create-large-nodes,~
9990   custom-line,~
9991   delimiters~(several~subkeys),~
9992   end-of-row,~
9993   first-col,~
9994   first-row,~
9995   hlines,~
9996   hvlines,~
9997   hvlines-except-borders,~
9998   last-col,~
9999   last-row,~
10000  left-margin,~
10001  light-syntax,~
10002  light-syntax-expanded,~
10003  matrix/columns-type,~
10004  no-cell-nodes,~
10005  notes~(several~subkeys),~
10006  nullify-dots,~
10007  pgf-node-code,~
10008  renew-dots,~
10009  renew-matrix,~
10010  respect-arraystretch,~
10011  rounded-corners,~
10012  right-margin,~
10013  rules~(with~the~subkeys~'color'~and~'width'),~
10014  small,~
10015  sub-matrix~(several~subkeys),~
10016  vlines,~
10017  xdots~(several~subkeys).
10018 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10019 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10020 {
10021   Unknown~key.\\
10022   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10023   \{NiceArray\}. \\
10024   That~key~will~be~ignored. \\
10025   \c_@@_available_keys_str
10026 }
10027 {
10028   The~available~keys~are~(in~alphabetic~order):~
10029   &~in~blocks,~
10030   ampersand~in~blocks,~
10031   b,~
10032   baseline,~
10033   c,~
10034   cell-space-bottom-limit,~
10035   cell-space-limits,~
10036   cell-space-top-limit,~
10037   code~after,~
10038   code~for~first~col,~

```

```

10039 code-for-first-row,~
10040 code-for-last-col,~
10041 code-for-last-row,~
10042 columns-width,~
10043 corners,~
10044 create-extra-nodes,~
10045 create-medium-nodes,~
10046 create-large-nodes,~
10047 extra-left-margin,~
10048 extra-right-margin,~
10049 first-col,~
10050 first-row,~
10051 hlines,~
10052 hvlines,~
10053 hvlines-except-borders,~
10054 last-col,~
10055 last-row,~
10056 left-margin,~
10057 light-syntax,~
10058 light-syntax-expanded,~
10059 name,~
10060 no-cell-nodes,~
10061 nullify-dots,~
10062 pgf-node-code,~
10063 renew-dots,~
10064 respect-arraystretch,~
10065 right-margin,~
10066 rounded-corners,~
10067 rules~(with~the~subkeys~'color'~and~'width'),~
10068 small,~
10069 t,~
10070 vlines,~
10071 xdots/color,~
10072 xdots/shorten-start,~
10073 xdots/shorten-end,~
10074 xdots/shorten-and~
10075 xdots/line-style.
10076 }

```

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`).

```

10077 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10078 {
10079   Unknown~key.\\
10080   The~key~'\l_keys_key_str'~is~unknown~for~the~
10081   \@@_full_name_env:. \\
10082   That~key~will~be~ignored. \\
10083   \c_@@_available_keys_str
10084 }
10085 {
10086   The~available~keys~are~(in~alphabetic~order):~
10087   &~in~blocks,~
10088   ampersand~in~blocks,~
10089   b,~
10090   baseline,~
10091   c,~
10092   cell-space-bottom-limit,~
10093   cell-space-limits,~
10094   cell-space-top-limit,~
10095   code-after,~
10096   code-for-first-col,~
10097   code-for-first-row,~
10098   code-for-last-col,~
10099   code-for-last-row,~

```

```

10100 columns-type,~
10101 columns-width,~
10102 corners,~
10103 create-extra-nodes,~
10104 create-medium-nodes,~
10105 create-large-nodes,~
10106 extra-left-margin,~
10107 extra-right-margin,~
10108 first-col,~
10109 first-row,~
10110 hlines,~
10111 hvlines,~
10112 hvlines-except-borders,~
10113 l,~
10114 last-col,~
10115 last-row,~
10116 left-margin,~
10117 light-syntax,~
10118 light-syntax-expanded,~
10119 name,~
10120 no-cell-nodes,~
10121 nullify-dots,~
10122 pgf-node-code,~
10123 r,~
10124 renew-dots,~
10125 respect-arraystretch,~
10126 right-margin,~
10127 rounded-corners,~
10128 rules~(with~the~subkeys~'color'~and~'width'),~
10129 small,~
10130 t,~
10131 vlines,~
10132 xdots/color,~
10133 xdots/shorten-start,~
10134 xdots/shorten-end,~
10135 xdots/shorten-and~
10136 xdots/line-style.
10137 }
10138 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10139 {
10140   Unknown~key.\\
10141   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10142   \{NiceTabular\}. \\
10143   That~key~will~be~ignored. \\
10144   \c_@@_available_keys_str
10145 }
10146 {
10147   The~available~keys~are~(in~alphabetic~order):~
10148   &~in~blocks,~
10149   ampersand~in~blocks,~
10150   b,~
10151   baseline,~
10152   c,~
10153   caption,~
10154   cell-space-bottom-limit,~
10155   cell-space-limits,~
10156   cell-space-top-limit,~
10157   code~after,~
10158   code~for~first~col,~
10159   code~for~first~row,~
10160   code~for~last~col,~
10161   code~for~last~row,~
10162   columns-width,~

```

```

10163     corners,~
10164     custom-line,~
10165     create-extra-nodes,~
10166     create-medium-nodes,~
10167     create-large-nodes,~
10168     extra-left-margin,~
10169     extra-right-margin,~
10170     first-col,~
10171     first-row,~
10172     hlines,~
10173     hvlines,~
10174     hvlines-except-borders,~
10175     label,~
10176     last-col,~
10177     last-row,~
10178     left-margin,~
10179     light-syntax,~
10180     light-syntax-expanded,~
10181     name,~
10182     no-cell-nodes,~
10183     notes~(several~subkeys),~
10184     nullify-dots,~
10185     pgf-node-code,~
10186     renew-dots,~
10187     respect-arraystretch,~
10188     right-margin,~
10189     rounded-corners,~
10190     rules~(with~the~subkeys~'color'~and~'width'),~
10191     short-caption,~
10192     t,~
10193     tabularnote,~
10194     vlines,~
10195     xdots/color,~
10196     xdots/shorten-start,~
10197     xdots/shorten-end,~
10198     xdots/shorten-and~
10199     xdots/line-style.
10200 }

10201 \@@_msg_new:nnn { Duplicate-name }
10202 {
10203     Duplicate~name.\\
10204     The-name-'l_keys_value_tl'-is~already~used~and~you~shouldn't~use~
10205     the~same~environment~name~twice.~You~can~go~on,~but,~
10206     maybe,~you~will~have~incorrect~results~especially~
10207     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10208     message~again,~use~the~key~'allow-duplicate-names'~in~
10209     '\token_to_str:N \NiceMatrixOptions'.\\
10210     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10211     { For~a~list~of~the~names~already~used,~type-H~<return>. }
10212 }
10213 {
10214     The-names~already~defined~in~this~document~are:~
10215     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10216 }

10217 \@@_msg_new:nn { Option~auto~for~columns-width }
10218 {
10219     Erroneous~use.\\
10220     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10221     That~key~will~be~ignored.
10222 }

10223 \@@_msg_new:nn { NiceTabularX~without~X }
10224 {
10225     NiceTabularX~without~X.\\

```

```
10226     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10227     However,~you~can~go~on.
10228   }
10229 \@@_msg_new:nn { Preamble~forgotten }
10230   {
10231     Preamble~forgotten.\\
10232     You~have~probably~forgotten~the~preamble~of~your~
10233     \@@_full_name_env:. \\
10234     This~error~is~fatal.
10235   }
```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	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 Tikz	113
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	126
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	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	208
38	About the package underscore	210
39	Error messages of the package	210