

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

January 20, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \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 7.0b of `nicematrix`, at the date of 2025/01/20.

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:NNTF \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 raised when the command `\rotate` is used with the key `c`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
332 {
```

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

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

```
335   \fi:
```

```
336 }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
350 \tl_new:N \l_@@_code_tl
```

For the key `pgf-node-code`. That code will be used when the nodes of the cells (that is to say the nodes of the form `i-j`) will be created.

```
351 \tl_new:N \l_@@_pgf_node_code_tl
```

The so-called `\CodeBefore` is splitted in two parts because we want to control the order of execution of some instructions.

```
352 \tl_new:N \g_@@_pre_code_before_tl
353 \tl_new:N \g_nicematrix_code_before_tl
```

The value of the key `code-before` will be added to the left of `\g_@@_pre_code_before_tl`. Idem for the code between `\CodeBefore` and `\Body`.

The so-called `\CodeAfter` is splitted in two parts because we want to control the order of execution of some instructions.

```
354 \tl_new:N \g_@@_pre_code_after_tl
355 \tl_new:N \g_nicematrix_code_after_tl
```

The `\CodeAfter` provided by the final user (with the key `code-after` or the keyword `\CodeAfter`) will be stored in the second token list.

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

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

```
357 \bool_new:N \l_@@_ampersand_bool
```

The counters `\l_@@_old_iRow_int` and `\l_@@_old_jCol_int` will be used to save the values of the potential LaTeX counters `iRow` and `jCol`. These LaTeX counters will be restored at the end of the environment.

```
358 \int_new:N \l_@@_old_iRow_int
359 \int_new:N \l_@@_old_jCol_int
```

The TeX counters `\c@iRow` and `\c@jCol` will be created in the beginning of `{NiceArrayWithDelims}` (if they don't exist previously).

The following sequence will contain the names (without backslash) of the commands created by `custom-line` by the key `command` or `ccommand` (commands used by the final user in order to draw horizontal rules).

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

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

```
361 \tl_new:N \l_@@_rules_color_tl
```

The sum of the weights of all the X-columns in the preamble. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

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

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

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

```
364 \dim_new:N \l_@@_X_columns_dim
```

This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if `\Hdotsfor` is used in that column.

```
365 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the `col` nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of `col` nodes).

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

It's possible to use the command `\NotEmpty` to specify explicitly that a cell must be considered as non empty by `nicematrix` (the Tikz nodes are constructed only in the non empty cells).

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

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

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

```
369 \bool_new:N \l_@@_code_before_bool
```

The following token list will contain the code inserted in each cell of the current row (this token list will be cleared at the beginning of each row).

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

385 \bool_new:N \g_@@_empty_cell_bool

```

The following dimensions will be used internally to compute the width of the potential “first column” and “last column”.

```

386 \dim_new:N \g_@@_width_last_col_dim
387 \dim_new:N \g_@@_width_first_col_dim

```

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces: `{imin}{jmin}{imax}{jmax}{options}{contents}`.

The variable is global because it will be modified in the cells of the array.

```

388 \seq_new:N \g_@@_blocks_seq

```

We also manage a sequence of the *positions* of the blocks. In that sequence, each block is represented by only five components: `{imin}{jmin}{imax}{jmax}{ name}`. A block with the key `hvlines` won’t appear in that sequence (otherwise, the lines in that block would not be drawn!).

```

389 \seq_new:N \g_@@_pos_of_blocks_seq

```

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

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

```

390 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

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

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

```

391 \seq_new:N \g_@@_pos_of_xdots_seq

```

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

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

```

392 \seq_new:N \g_@@_pos_of_stroken_blocks_seq

```

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

```

393 \clist_new:N \l_@@_corners_cells_clist

```

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

```

394 \seq_new:N \g_@@_submatrix_names_seq

```

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

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

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

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

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

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

The following counters will be used when drawing the rules.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


Variables for the exterior rows and columns

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

- **First row**

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

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

- **First column**

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

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

- **Last row**

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

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

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

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

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

```

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

```

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

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

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

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

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

Some utilities

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

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

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

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

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

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

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

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

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

The following internal parameters are for:

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

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

5 The command `\tabularnote`

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

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

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

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

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

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

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

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

```
485 \seq_new:N \g_@@_notes_in_caption_seq
```

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

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

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

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

```
488 \newcounter { nicematrix_draft }
```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

$$\{label\}\{text\ of\ the\ tabularnote\}.$$

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

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

```

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

```

```

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

```

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

```

589     \hbox_set:Nn \l_tmpa_box
590     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

646 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
647   { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }

```

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

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

```

7 The options

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

```

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

```


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

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

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

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

The following dimensions correspond to the options `cell-space-top-limit` and `co` (these parameters are inspired by the package `cellspace`).

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

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

```
702 \bool_new:N \l_@@_xdots_h_labels_bool
```

The following dimension is the distance between two dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.45 em but it will be changed if the option `small` is used.

```
703 \dim_new:N \l_@@_xdots_inter_dim
704 \hook_gput_code:nnn { begindocument } { . }
705 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }
```

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the distance between a node (in fact an anchor of that node) and a dotted line (for real dotted lines, the actual distance may, of course, be a bit larger, depending of the exact position of the dots).

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

The unit is `em` and that's why we fix the dimension after the preamble.

The following dimension is the radius of the dots for the dotted lines (when `line-style` is equal to `standard`, which is the initial value). The initial value is 0.53 pt but it will be changed if the option `small` is used.

```
713 \dim_new:N \l_@@_xdots_radius_dim
714 \hook_gput_code:nnn { begindocument } { . }
715 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

The unit is `em` and that's why we fix the dimension after the preamble.

The token list `\l_@@_xdots_line_style_tl` corresponds to the option `tikz` of the commands `\Cdots`, `\Ldots`, etc. and of the options `line-style` for the environments and `\NiceMatrixOptions`. The constant `\c_@@_standard_tl` will be used in some tests.

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

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax` and the boolean `\l_@@_light_syntax_expanded_bool` correspond to the the option `light-syntax-expanded`.

```
719 \bool_new:N \l_@@_light_syntax_bool
720 \bool_new:N \l_@@_light_syntax_expanded_bool
```

The string `\l_@@_baseline_tl` may contain one of the three values `t`, `c` or `b` as in the option of the environment `{array}`. However, it may also contain **an integer** (which represents the number of the row to which align the array).

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

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

```
723 \bool_new:N \l_@@_amp_in_blocks_bool
```

The flag `\l_@@_exterior_arraycolsep_bool` corresponds to the option `exterior-arraycolsep`. If this option is set, a space equal to `\arraycolsep` will be put on both sides of an environment `{NiceArray}` (as it is done in `{array}` of `array`).

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

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

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

The following parameter correspond to the key `corners`. The elements of that `clist` must be within `NW`, `SW`, `NE` and `SE`.

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

```
728 \dim_new:N \l_@@_notes_above_space_dim
729 \hook_gput_code:nnn { begindocument } { . }
730 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

We use a hook only by security in case `revtex4-1` is used (even though it is obsolete).

The flag `\l_@@_nullify_dots_bool` corresponds to the option `nullify-dots`. When the flag is down, the instructions like `\vdots` are inserted within a `\hphantom` (and so the constructed matrix has exactly the same size as a matrix constructed with the classical `{matrix}` and `\ldots`, `\vdots`, etc.).

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

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

```
732 \cs_new_protected:Npn \@@_reset_arraystretch:
733 { \cs_set_nopar:Npn \arraystretch { 1 } }
```

The following flag will be used when the current options specify that all the columns of the array must have the same width equal to the largest width of a cell of the array (except the cells of the potential exterior columns).

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

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

```
735 \bool_new:N \g_@@_recreate_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.

```
736 \str_new:N \l_@@_name_str
```

The boolean `\l_@@_medium_nodes_bool` will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

```
737 \bool_new:N \l_@@_medium_nodes_bool
738 \bool_new:N \l_@@_large_nodes_bool
```

The boolean `\l_@@_except_borders_bool` will be raised when the key `hvlines-except-borders` will be used (but that key has also other effects).

```
739 \bool_new:N \l_@@_except_borders_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

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

The dimensions `\l_@@_extra_left_margin_dim` and `\l_@@_extra_right_margin_dim` correspond to the options `extra-left-margin` and `extra-right-margin`.

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

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
744 \tl_new:N \l_@@_end_of_row_tl
745 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

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

```

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

```

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

```

The options `c`, `t` and `b` of the environment `{NiceArray}` have the same meaning as the option of the classical environment `{array}`.

```

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

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```

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

```

We test whether we are in the measuring phase of an environment of `amsmath` (always loaded by `nicematrix`) because we want to avoid a fallacious message of duplicate name in this case.

```

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

```

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

```

989 \keys_define:nn { nicematrix }
990 {

```

```

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

```

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

```

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

```

With the option `renew-matrix`, the environment `{matrix}` of `amsmath` and its variants are redefined to behave like the environment `{NiceMatrix}` and its variants.

```

1047   renew-matrix .code:n = \@@_renew_matrix: ,
1048   renew-matrix .value_forbidden:n = true ,

```


The option `exterior-arraycolsep` will have effect only in `{NiceArray}` for those who want to have for `{NiceArray}` the same behaviour as `{array}`.

```
1049 exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
```

If the option `columns-width` is used, all the columns will have the same width. In `\NiceMatrixOptions`, the special value `auto` is not available.

```
1050 columns-width .code:n =
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
1051 \str_if_eq:eeTF { #1 } { auto }
1052 { \@@_error:n { Option~auto~for~columns~width } }
1053 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

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

`\NiceMatrixOptions` is the command of the `nicematrix` package to fix options at the document level. The scope of these specifications is the current TeX group.

```
1067 \NewDocumentCommand \NiceMatrixOptions { m }
1068 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }
```

We finalise the definition of the set of keys “`nicematrix / NiceMatrix`”. That set of keys will be used by `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

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

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

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

In the environments {NiceArray} and its variants, the option last-col must be used without value because the number of columns of the array is read from the preamble of the array.

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

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

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

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

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

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

The `\CodeAfter` (inserted with the key `code-after` or after the keyword `\CodeAfter`) may always begin with a list of pairs `key=value` between square brackets. Here is the corresponding set of keys. We *must* put the following instructions *after* the :

```
CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix

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

8 Important code used by `{NiceArrayWithDelims}`

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

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

`\g_@@_cell_after_hook_tl` will be set during the composition of the box `\l_@@_cell_box` and will be used *after* the composition in order to modify that box.

```
1155   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1157   \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don't do this incrementation in the `\everycr` because some packages, like `arydshln`, create special rows in the `\halign` that we don't want to take into account.

```
1158   \int_compare:nNnT \c@jCol = \c_one_int
1159     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }
```

The content of the cell is composed in the box `\l_@@_cell_box`. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` is in the `\@@_cell_end:`.

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

The following command is nullified in the tabulars.

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

The following command will be nullified unless there is a first row. Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT \c@iRow
  {
    \int_compare:nNnT \c@jCol > 0
    {
      \l_@@_code_for_first_row_tl
      \xglobal \colorlet { nicematrix-first-row } { . }
    }
  }
}

```

We will use a version a little more efficient.

```

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

```

The following command will be nullified unless there is a last row and we know its value (*ie*: `\l_@@_lat_row_int > 0`).

```

\cs_new_protected:Npn \@@_tuning_last_row:
{
  \int_compare:nNnT \c@iRow = \l_@@_last_row_int
  {
    \l_@@_code_for_last_row_tl
    \xglobal \colorlet { nicematrix-last-row } { . }
  }
}

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

The following commands are nullified in the tabulars.

```

1183 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1184 {
1185   \m@th % added 2024/11/21
1186   \c_math_toggle_token

```

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

The token list `\g_@@_cell_after_hook_tl` is (potentially) set during the composition of the box `\l_@@_cell_box` and is used now *after* the composition in order to modify that box.

```

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

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

1289     \@@_update_max_cell_width:

```

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

```

1290     \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

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

```


I don't know why the following 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.

```

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

```

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

```

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

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

```

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

```

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

```

1422     \@tabarray

```

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

```

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

```

We keep in memory the standard version of \ialign because we will redefine \ialign in the environment {NiceArrayWithDelims} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), array uses \ar@ialign instead of \ialign. In that case, of course, you do a saving of \ar@ialign.

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

The counter `\c@iRow` has the value `-1` only if there is a “first row” and that we are before that “first row”, i.e. just before the beginning of the array.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

1535 \cs_new_protected:Npn \@_pre_array_ii:
1536 {

```

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

```

1537   \int_gzero:N \g_@@_total_X_weight_int
1538   \@_expand_clist:N \l_@@_hlines_clist
1539   \@_expand_clist:N \l_@@_vlines_clist
1540   \@_patch_booktabs:
1541   \box_clear_new:N \l_@@_cell_box
1542   \normalbaselines

```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```

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

```

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}`.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```

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

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the aux file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

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

```

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

```

1635 \seq_gclear:N \g_@@_multicolumn_cells_seq
1636 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

The counter `\c@iRow` will be used to count the rows of the array (its incrementation will be in the first cell of the row).

```

1637 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1638 \int_gzero_new:N \g_@@_row_total_int

```

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

```

1639 \int_gzero_new:N \g_@@_col_total_int
1640 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1641 \bool_gset_false:N \g_@@_last_col_found_bool

```

During the construction of the array, the instructions `\Cdots`, `\Ldots`, etc. will be written in token lists `\g_@@_Cdots_lines_tl`, etc. which will be executed after the construction of the array.

```

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

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

```

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

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

```

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

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

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

```

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

```

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

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

```

Now the `\CodeBefore`.

```

1680   \bool_if:NT \l_@@_code_before_bool \@@_exec_code_before:

```

The value of `\g_@@_pos_of_blocks_seq` has been written on the aux file and loaded before the (potential) execution of the `\CodeBefore`. Now, we clear that variable because it will be reconstructed during the creation of the array.

```

1681   \seq_gset_eq:NN \g_@@_pos_of_blocks_seq \g_@@_future_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

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

```

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

```

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

```

The value -2 is important.

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

```

1685   \@@_pre_array_ii:

```

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```

1686   \box_clear_new:N \l_@@_the_array_box

```

We compute the width of both delimiters. We remind that, when the environment `{NiceArray}` is used, it’s possible to specify the delimiters in the preamble (eg `[ccc]`).

```

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

```

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

```

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

```


Here is the beginning of the box which will contain the array. The `\hbox_set_end:` corresponding to this `\hbox_set:Nw` will be in the second part of the environment (and the closing `\c_math_toggle_token` also).

```

1701   \hbox_set:Nw \l_@@_the_array_box
1702   \skip_horizontal:N \l_@@_left_margin_dim
1703   \skip_horizontal:N \l_@@_extra_left_margin_dim
1704   \bool_if:NT \c_@@_recent_array_bool
1705     { \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).

```

1706   \m@th
1707   \c_math_toggle_token
1708   \bool_if:NTF \l_@@_light_syntax_bool
1709     { \use:c { @@-light-syntax } }
1710     { \use:c { @@-normal-syntax } }
1711 }

```

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

```

1712 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1713 {
1714   \tl_set:Nn \l_tmpa_tl { #1 }
1715   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1716     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1717   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1718   \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...

```

1719   \@@_pre_array:
1720 }

```

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).

```

1721 \cs_new_protected:Npn \@@_pre_code_before:
1722 {

```

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).

```

1723   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1724   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1725   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1726   \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.

```

1727   \pgfsys@markposition { \@@_env: - position }
1728   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1729   \pgfpicture
1730   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1731 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1732 {
1733   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1734   \pgfcoordinate { \@@_env: - row - ##1 }
1735   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1736 }

```

Now, the recreation of the col nodes.

```

1737 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1738 {
1739   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1740   \pgfcoordinate { \@@_env: - col - ##1 }
1741   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1742 }

```

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

```

1743 \@@_create_diag_nodes:

```

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

```

1744 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1745 \endpgfpicture

```

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

```

1746 \@@_create_blocks_nodes:
1747 \IfPackageLoadedT { tikz }
1748 {
1749   \tikzset
1750   {
1751     every-picture / .style =
1752     { overlay , name-prefix = \@@_env: - }
1753   }
1754 }
1755 \cs_set_eq:NN \cellcolor \@@_cellcolor
1756 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1757 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1758 \cs_set_eq:NN \rowcolor \@@_rowcolor
1759 \cs_set_eq:NN \rowcolors \@@_rowcolors
1760 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1761 \cs_set_eq:NN \arraycolor \@@_arraycolor
1762 \cs_set_eq:NN \columncolor \@@_columncolor
1763 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1764 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1765 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1766 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1767 \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
1768 \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
1769 }

```

```

1770 \cs_new_protected:Npn \@@_exec_code_before:
1771 {

```

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...

```

1772 \clist_map_inline:Nn \l_@@_corners_cells_clist
1773 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1774 \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`.

```
1775 \@@_add_to_colors_seq:n { { nocolor } } { }
1776 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1777 \group_begin:
```

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

```
1778 \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`).

```
1779 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1780 { \@@_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`.

```
1781 \exp_last_unbraced:No \@@_CodeBefore_keys:
1782 \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.

```
1783 \@@_actually_color:
1784 \l_@@_code_before_tl
1785 \q_stop
1786 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1787 \group_end:
1788 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1789 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1790 }
```

```
1791 \keys_define:n { nicematrix / CodeBefore }
1792 {
1793   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1794   create-cell-nodes .default:n = true ,
1795   sub-matrix .code:n = \keys_set:n { nicematrix / sub-matrix } { #1 } ,
1796   sub-matrix .value_required:n = true ,
1797   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1798   delimiters / color .value_required:n = true ,
1799   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1800 }
1801 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1802 {
1803   \keys_set:n { nicematrix / CodeBefore } { #1 }
1804   \@@_CodeBefore:w
1805 }
```

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.

```
1806 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1807 {
1808   \bool_if:NT \g_@@_aux_found_bool
1809   {
1810     \@@_pre_code_before:
1811     #1
```

```

1812 }
1813 }

```

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.

```

1814 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1815 {
1816   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1817   {
1818     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1819     \pgfcoordinate { \@@_env: - row - ##1 - base }
1820     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1821     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1822     {
1823       \cs_if_exist:cT
1824       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1825       {
1826         \pgfsys@getposition
1827         { \@@_env: - ##1 - #####1 - NW }
1828         \@@_node_position:
1829         \pgfsys@getposition
1830         { \@@_env: - ##1 - #####1 - SE }
1831         \@@_node_position_i:
1832         \@@_pgf_rect_node:nnn
1833         { \@@_env: - ##1 - #####1 }
1834         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1835         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1836       }
1837     }
1838   }
1839   \int_step_inline:nn \c@iRow
1840   {
1841     \pgfnodealias
1842     { \@@_env: - ##1 - last }
1843     { \@@_env: - ##1 - \int_use:N \c@jCol }
1844   }
1845   \int_step_inline:nn \c@jCol
1846   {
1847     \pgfnodealias
1848     { \@@_env: - last - ##1 }
1849     { \@@_env: - \int_use:N \c@iRow - ##1 }
1850   }
1851   \@@_create_extra_nodes:
1852 }

1853 \cs_new_protected:Npn \@@_create_blocks_nodes:
1854 {
1855   \pgfpicture
1856   \pgf@relevantforpicturesizefalse
1857   \pgfrememberpicturepositiononpagetrue
1858   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1859   { \@@_create_one_block_node:nnnnn ##1 }
1860   \endpgfpicture
1861 }

```

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

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).

```

1862 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1863 {
1864   \tl_if_empty:nF { #5 }
1865   {
1866     \@@_qpoint:n { col - #2 }
1867     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1868     \@@_qpoint:n { #1 }
1869     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1870     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1871     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1872     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1873     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1874     \@@_pgf_rect_node:nnnnn
1875     { \@@_env: - #5 }
1876     { \dim_use:N \l_tmpa_dim }
1877     { \dim_use:N \l_tmpb_dim }
1878     { \dim_use:N \l_@@_tmpc_dim }
1879     { \dim_use:N \l_@@_tmpd_dim }
1880   }
1881 }

1882 \cs_new_protected:Npn \@@_patch_for_revtex:
1883 {
1884   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1885   \cs_set_eq:NN \@array \@array@array
1886   \cs_set_eq:NN \@tabular \@tabular@array
1887   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1888   \cs_set_eq:NN \array \array@array
1889   \cs_set_eq:NN \endarray \endarray@array
1890   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1891   \cs_set_eq:NN \@mkpream \@mkpream@array
1892   \cs_set_eq:NN \@classx \@classx@array
1893   \cs_set_eq:NN \insert@column \insert@column@array
1894   \cs_set_eq:NN \@arraycr \@arraycr@array
1895   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1896   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1897 }

```

10 The environment `{NiceArrayWithDelims}`

```

1898 \NewDocumentEnvironment { NiceArrayWithDelims }
1899 { m m 0 { } m ! 0 { } t \CodeBefore }
1900 {
1901   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1902   \@@_provide_pgfsyspdfmark:
1903   \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 expositant to a matrix in a mathematical formula.

```

1904   \bgroup

1905   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1906   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1907   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1908   \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1909   \int_gzero:N \g_@@_block_box_int
1910   \dim_zero:N \g_@@_width_last_col_dim

```

```

1911 \dim_zero:N \g_@@_width_first_col_dim
1912 \bool_gset_false:N \g_@@_row_of_col_done_bool
1913 \str_if_empty:NT \g_@@_name_env_str
1914 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1915 \bool_if:NTF \l_@@_tabular_bool
1916 \mode_leave_vertical:
1917 \@@_test_if_math_mode:
1918 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1919 \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.

```

1920 \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.

```

1921 \cs_if_exist:NT \tikz@library@external@loaded
1922 {
1923 \tikzexternaldisable
1924 \cs_if_exist:NT \ifstandalone
1925 { \tikzset { external / optimize = false } }
1926 }

```

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

```

1927 \int_gincr:N \g_@@_env_int
1928 \bool_if:NF \l_@@_block_auto_columns_width_bool
1929 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1930 \seq_gclear:N \g_@@_blocks_seq
1931 \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`.

```

1932 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1933 \seq_gclear:N \g_@@_pos_of_xdots_seq
1934 \tl_gclear_new:N \g_@@_code_before_tl
1935 \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.

```

1936 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1937 {
1938 \bool_gset_true:N \g_@@_aux_found_bool
1939 \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1940 }
1941 { \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.

```

1942 \tl_gclear:N \g_@@_aux_tl
1943 \tl_if_empty:NF \g_@@_code_before_tl
1944 {
1945 \bool_set_true:N \l_@@_code_before_bool
1946 \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1947 }

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1948 \tl_if_empty:NF \g_@@_pre_code_before_tl
1949 { \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`.

```

1950 \bool_if:NTF \g_@@_delims_bool
1951 { \keys_set:nn { nicematrix / pNiceArray } }
1952 { \keys_set:nn { nicematrix / NiceArray } }
1953 { #3 , #5 }

```

```

1954 \@@_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:`.

```

1955 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1956 }

```

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

```

1957 {
1958 \bool_if:NTF \l_@@_light_syntax_bool
1959 { \use:c { end @@-light-syntax } }
1960 { \use:c { end @@-normal-syntax } }
1961 \c_math_toggle_token
1962 \skip_horizontal:N \l_@@_right_margin_dim
1963 \skip_horizontal:N \l_@@_extra_right_margin_dim
1964
1965 % awful workaround
1966 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1967 {
1968 \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1969 {
1970 \skip_horizontal:N - \l_@@_columns_width_dim
1971 \bool_if:NTF \l_@@_tabular_bool
1972 { \skip_horizontal:n { - 2 \tabcolsep } }
1973 { \skip_horizontal:n { - 2 \arraycolsep } }
1974 }
1975 }
1976 \hbox_set_end:
1977 \bool_if:NT \c_@@_recent_array_bool
1978 { \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.

```

1979 \bool_if:NT \l_@@_width_used_bool
1980 {
1981 \int_if_zero:nT \g_@@_total_X_weight_int
1982 { \@@_error_or_warning:n { width-without-X-columns } }
1983 }

```

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`.

```

1984 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1985 { \@@_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).

```

1986 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1987 {
1988   \bool_if:NF \l_@@_last_row_without_value_bool
1989   {
1990     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1991     {
1992       \@@_error:n { Wrong~last~row }
1993       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1994     }
1995   }
1996 }

```

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

```

1997 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1998 \bool_if:NTF \g_@@_last_col_found_bool
1999 { \int_gdecr:N \c@jCol }
2000 {
2001   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2002   { \@@_error:n { last~col~not~used } }
2003 }

```

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

```

2004 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2005 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2006 \int_if_zero:nT \l_@@_first_col_int
2007 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2008 \bool_if:nTF { ! \g_@@_delims_bool }
2009 {
2010   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2011   \@@_use_arraybox_with_notes_c:
2012   {
2013     \str_if_eq:eeTF \l_@@_baseline_tl { b }
2014     \@@_use_arraybox_with_notes_b:
2015     \@@_use_arraybox_with_notes:
2016   }
2017 }

```

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).

```

2018 {
2019   \int_if_zero:nTF \l_@@_first_row_int
2020   {
2021     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2022     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2023   }
2024   { \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

```

2025 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2026 {
2027   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2028   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim

```

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

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


```

2029     }
2030     { \dim_zero:N \l_tmpb_dim }
2031 \hbox_set:Nn \l_tmpa_box
2032 {
2033     \m@th % added 2024/11/21
2034     \c_math_toggle_token
2035     \@@_color:o \l_@@_delimiters_color_tl
2036     \exp_after:wN \left \g_@@_left_delim_tl
2037     \vcenter
2038     {

```

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.

```

2039         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2040     \hbox
2041     {
2042         \bool_if:NTF \l_@@_tabular_bool
2043         { \skip_horizontal:N -\tabcolsep }
2044         { \skip_horizontal:N -\arraycolsep }
2045     \@@_use_arraybox_with_notes_c:
2046     \bool_if:NTF \l_@@_tabular_bool
2047     { \skip_horizontal:N -\tabcolsep }
2048     { \skip_horizontal:N -\arraycolsep }
2049     }

```

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

```

2050         \skip_vertical:N -\l_tmpb_dim
2051         \skip_vertical:N \arrayrulewidth
2052     }
2053     \exp_after:wN \right \g_@@_right_delim_tl
2054     \c_math_toggle_token
2055 }

```

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.

```

2056     \bool_if:NTF \l_@@_delimiters_max_width_bool
2057     {
2058         \@@_put_box_in_flow_bis:nn
2059         \g_@@_left_delim_tl
2060         \g_@@_right_delim_tl
2061     }
2062     \@@_put_box_in_flow:
2063 }

```

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

```

2064     \bool_if:NT \g_@@_last_col_found_bool
2065     { \skip_horizontal:N \g_@@_width_last_col_dim }
2066     \bool_if:NT \l_@@_preamble_bool
2067     {
2068         \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2069         { \@@_warning_gredirect_none:n { columns-not-used } }
2070     }
2071     \@@_after_array:

```

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

```

2072     \egroup

```

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

```

2073     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2074     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2075     \iow_now:Ne \@mainaux
2076     {

```

```

2077     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ t1 }
2078     { \exp_not:o \g_@@_aux_t1 }
2079   }
2080   \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2081   \bool_if:NT \g_@@_footnote_bool \endsavenotes
2082 }

```

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 .

```

2083 \cs_new_protected:Npn \@@_compute_width_X:
2084 {
2085   \tl_gput_right:Ne \g_@@_aux_t1
2086   {
2087     \bool_set_true:N \l_@@_X_columns_aux_bool
2088     \dim_set:Nn \l_@@_X_columns_dim
2089     {
2090       \dim_compare:nNnTF
2091       {
2092         \dim_abs:n
2093         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2094       }
2095       <
2096       { 0.001 pt }
2097       { \dim_use:N \l_@@_X_columns_dim }
2098     }
2099     \dim_eval:n
2100     {
2101       ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2102       / \int_use:N \g_@@_total_X_weight_int
2103       + \l_@@_X_columns_dim
2104     }
2105   }
2106 }
2107 }
2108 }

```

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_t1`. The modified version will be stored in `\g_@@_array_preamble_t1` also.

```

2109 \cs_new_protected:Npn \@@_transform_preamble:
2110 {
2111   \@@_transform_preamble_i:
2112   \@@_transform_preamble_ii:
2113 }
2114 \cs_new_protected:Npn \@@_transform_preamble_i:
2115 {
2116   \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`).

```
2117 \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.

```
2118 \bool_gset_false:N \g_tmpb_bool
```

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

```
2119 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2120 \int_zero:N \l_tmpa_int
2121 \tl_gclear:N \g_@@_array_preamble_tl
2122 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2123 {
2124   \tl_gset:Nn \g_@@_array_preamble_tl
2125     { ! { \skip_horizontal:N \arrayrulewidth } }
2126 }
2127 {
2128   \clist_if_in:NnT \l_@@_vlines_clist 1
2129   {
2130     \tl_gset:Nn \g_@@_array_preamble_tl
2131       { ! { \skip_horizontal:N \arrayrulewidth } }
2132   }
2133 }
```

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

```
2134 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2135 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2136 \@@_replace_columncolor:
2137 }
```

```
2138 \hook_gput_code:nnn { begindocument } { . }
2139 {
2140   \IfPackageLoadedTF { colortbl }
2141   {
```

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).

```
2142   \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2143   \cs_new_protected:Npn \@@_replace_columncolor:
2144     {
2145       \regex_replace_all:NnN
2146         \c_@@_columncolor_regex
2147         { \c { @@_columncolor_preamble } }
2148         \g_@@_array_preamble_tl
2149     }
2150   }
2151   {
2152     \cs_new_protected:Npn \@@_replace_columncolor:
2153       { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2154   }
2155 }
```

```
2156 \cs_new_protected:Npn \@@_transform_preamble_ii:
2157 {
```

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

```

2158     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2159     {
2160         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2161         { \bool_gset_true:N \g_@@_delims_bool }
2162     }
2163     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2164     \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).

```

2165     \int_if_zero:nTF \l_@@_first_col_int
2166     { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2167     {
2168         \bool_if:NF \g_@@_delims_bool
2169         {
2170             \bool_if:NF \l_@@_tabular_bool
2171             {
2172                 \clist_if_empty:NT \l_@@_vlines_clist
2173                 {
2174                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2175                     { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2176                 }
2177             }
2178         }
2179     }
2180     \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2181     { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2182     {
2183         \bool_if:NF \g_@@_delims_bool
2184         {
2185             \bool_if:NF \l_@@_tabular_bool
2186             {
2187                 \clist_if_empty:NT \l_@@_vlines_clist
2188                 {
2189                     \bool_if:NF \l_@@_exterior_arraycolsep_bool
2190                     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2191                 }
2192             }
2193         }
2194     }

```

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`).

```

2195     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2196     {

```

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.

```

2197         \bool_if:NF \c_@@_testphase_table_bool
2198         {
2199             \tl_gput_right:Nn \g_@@_array_preamble_tl
2200             { > { \@@_error_too_much_cols: } 1 }
2201         }
2202     }
2203 }

```

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.

```

2204 \cs_new_protected:Npn \@@_rec_preamble:n #1
2205 {

```

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

```

2206     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2207     { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2208     {

```

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

```

2209     \cs_if_exist:cTF { NC @ find @ #1 }
2210     {
2211         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2212         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2213     }
2214     {
2215         \str_if_eq:nnTF { #1 } { S }
2216         { \@@_fatal:n { unknown~column~type~S } }
2217         { \@@_fatal:nn { unknown~column~type } { #1 } }
2218     }
2219 }
2220 }

```

For c, l and r

```

2221 \cs_new_protected:Npn \@@_c #1
2222 {
2223     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2224     \tl_gclear:N \g_@@_pre_cell_tl
2225     \tl_gput_right:Nn \g_@@_array_preamble_tl
2226     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2227     \int_gincr:N \c@jCol
2228     \@@_rec_preamble_after_col:n
2229 }

2230 \cs_new_protected:Npn \@@_l #1
2231 {
2232     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2233     \tl_gclear:N \g_@@_pre_cell_tl
2234     \tl_gput_right:Nn \g_@@_array_preamble_tl
2235     {
2236         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2237         l
2238         < \@@_cell_end:
2239     }
2240     \int_gincr:N \c@jCol
2241     \@@_rec_preamble_after_col:n
2242 }

2243 \cs_new_protected:Npn \@@_r #1
2244 {
2245     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2246     \tl_gclear:N \g_@@_pre_cell_tl
2247     \tl_gput_right:Nn \g_@@_array_preamble_tl
2248     {
2249         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2250         r
2251         < \@@_cell_end:

```

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`.

```

2252     }
2253     \int_gincr:N \c@jCol
2254     \@@_rec_preamble_after_col:n
2255 }

```

For ! and @

```

2256 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2257 {
2258     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2259     \@@_rec_preamble:n
2260 }
2261 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2262 \cs_new_protected:cpn { @@ _ | } #1
2263 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2264     \int_incr:N \l_tmpa_int
2265     \@@_make_preamble_i_i:n
2266 }

```

```

2267 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2268 {
2269     \str_if_eq:nnTF { #1 } { | }
2270     { \use:c { @@ _ | } | }
2271     { \@@_make_preamble_i_ii:nn { } #1 }
2272 }

```

```

2273 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2274 {
2275     \str_if_eq:nnTF { #2 } { [ ]
2276     { \@@_make_preamble_i_ii:nw { #1 } [ ]
2277     { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2278 }

```

```

2279 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2280 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

```

```

2281 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2282 {
2283     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2284     \tl_gput_right:Ne \g_@@_array_preamble_tl
2285     {

```

Here, the command \dim_use:N is mandatory.

```

2286         \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2287     }
2288     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2289     {
2290         \@@_vline:n
2291         {
2292             position = \int_eval:n { \c@jCol + 1 } ,
2293             multiplicity = \int_use:N \l_tmpa_int ,
2294             total-width = \dim_use:N \l_@@_rule_width_dim ,
2295             #2
2296         }

```

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.

```

2297     }
2298     \int_zero:N \l_tmpa_int
2299     \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2300     \@@_rec_preamble:n #1
2301 }

```

```

2302 \cs_new_protected:cpn { @@ _ > } #1 #2
2303 {
2304   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2305   \@@_rec_preamble:n
2306 }
2307 \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.

```

2308 \keys_define:nn { nicematrix / p-column }
2309 {
2310   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2311   r .value_forbidden:n = true ,
2312   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2313   c .value_forbidden:n = true ,
2314   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2315   l .value_forbidden:n = true ,
2316   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2317   S .value_forbidden:n = true ,
2318   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2319   p .value_forbidden:n = true ,
2320   t .meta:n = p ,
2321   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2322   m .value_forbidden:n = true ,
2323   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2324   b .value_forbidden:n = true
2325 }

```

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

```

2326 \cs_new_protected:Npn \@@_p #1
2327 {
2328   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2329   \@@_make_preamble_ii_i:n
2330 }
2331 \cs_set_eq:NN \@@_b \@@_p
2332 \cs_set_eq:NN \@@_m \@@_p
2333 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2334 {
2335   \str_if_eq:nnTF { #1 } { [ ]
2336     { \@@_make_preamble_ii_ii:w [ ]
2337       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2338   }
2339   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2340   { \@@_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.

```

2341 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2342 {

```

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).

```

2343   \str_set:Nn \l_@@_hpos_col_str { j }
2344   \@@_keys_p_column:n { #1 }
2345   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2346 }
2347 \cs_new_protected:Npn \@@_keys_p_column:n #1
2348 { \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.

```

2349 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2350 {
2351   \use:e
2352   {
2353     \@@_make_preamble_ii_v:nnnnnnnn
2354     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2355     { \dim_eval:n { #1 } }
2356     {

```

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.

```

2357       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2358       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2359       {

```

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

```

2360         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2361         { \str_lowercase:o \l_@@_hpos_col_str }
2362     }
2363     \IfPackageLoadedTF { ragged2e }
2364     {
2365       \str_case:on \l_@@_hpos_col_str
2366       {
2367         c { \exp_not:N \Centering }
2368         l { \exp_not:N \RaggedRight }
2369         r { \exp_not:N \RaggedLeft }
2370       }
2371     }
2372     {
2373       \str_case:on \l_@@_hpos_col_str
2374       {
2375         c { \exp_not:N \centering }
2376         l { \exp_not:N \raggedright }
2377         r { \exp_not:N \raggedleft }
2378       }
2379     }
2380     #3
2381   }
2382   { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2383   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2384   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2385   { #2 }
2386   {
2387     \str_case:onF \l_@@_hpos_col_str
2388     {
2389       { j } { c }
2390       { si } { c }
2391     }

```

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

```

2392       { \str_lowercase:o \l_@@_hpos_col_str }
2393     }
2394   }

```

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

```

2395     \int_gincr:N \c@jCol
2396     \@@_rec_preamble_after_col:n
2397   }

```


#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*.

```

2398 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2399 {
2400   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2401   {
2402     \tl_gput_right:Nn \g_@@_array_preamble_tl
2403     { > \@@_test_if_empty_for_S: }
2404   }
2405   { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
2406   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2407   \tl_gclear:N \g_@@_pre_cell_tl
2408   \tl_gput_right:Nn \g_@@_array_preamble_tl
2409   {
2410     > {

```

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.

```

2411   \dim_set:Nn \l_@@_col_width_dim { #2 }
2412   \bool_if:NT \c_@@_testphase_table_bool
2413   { \tag_struct_begin:n { tag = Div } }
2414   \@@_cell_begin:

```

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

```

2415   \use:c { #7 } [ #1 ] { #2 }

```

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

```

2416   \everypar
2417   {
2418     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2419     \everypar { }
2420   }
2421   \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.).

```

2422   #3

```

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

```

2423   \g_@@_row_style_tl
2424   \arraybackslash
2425   #5
2426   }
2427   #8
2428   < {
2429   #6

```

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

```

2430   \@finalstrut \@arstrutbox
2431   \use:c { end #7 }

```

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

```

2432         #4
2433         \@@_cell_end:
2434         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2435     }
2436 }
2437 }

```

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

```

2438 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2439 {

```

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 `&`).

```

2440     \group_align_safe_begin:
2441     \peek_meaning:NTF &
2442     \@@_the_cell_is_empty:
2443     {
2444         \peek_meaning:NTF \
2445         \@@_the_cell_is_empty:
2446         {
2447             \peek_meaning:NTF \crcr
2448             \@@_the_cell_is_empty:
2449             \group_align_safe_end:
2450         }
2451     }
2452 }

2453 \cs_new_protected:Npn \@@_the_cell_is_empty:
2454 {
2455     \group_align_safe_end:
2456     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2457     {

```

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

```

2458         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2459         \skip_horizontal:N \l_@@_col_width_dim
2460     }
2461 }

2462 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2463 {
2464     \peek_meaning:NT \__siunitx_table_skip:n
2465     { \bool_gset_true:N \g_@@_empty_cell_bool }
2466 }

```

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.

```

2467 \cs_new_protected:Npn \@@_center_cell_box:
2468 {

```

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

```

2469     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2470     {
2471         \int_compare:nNnT
2472         { \box_ht:N \l_@@_cell_box }
2473         >

```

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).

```

2474     { \box_ht:N \strutbox }
2475     {
2476       \hbox_set:Nn \l_@@_cell_box
2477       {
2478         \box_move_down:nn
2479         {
2480           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2481             + \baselineskip ) / 2
2482         }
2483         { \box_use:N \l_@@_cell_box }
2484       }
2485     }
2486   }
2487 }

```

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

```

2488 \cs_new_protected:Npn \@@_V #1 #2
2489 {
2490   \str_if_eq:nnTF { #1 } { [ ]
2491     { \@@_make_preamble_V_i:w [ ]
2492       { \@@_make_preamble_V_i:w [ ] { #2 } }
2493     }
2494   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2495     { \@@_make_preamble_V_ii:nn { #1 } }
2496   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2497   {
2498     \str_set:Nn \l_@@_vpos_col_str { p }
2499     \str_set:Nn \l_@@_hpos_col_str { j }
2500     \@@_keys_p_column:n { #1 }
2501     \IfPackageLoadedTF { varwidth }
2502       { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2503       {
2504         \@@_error_or_warning:n { varwidth-not-loaded }
2505         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2506       }
2507   }

```

For w and W

```

2508 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2509 \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.

```

2510 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2511 {
2512   \str_if_eq:nnTF { #3 } { s }
2513     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2514     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2515 }

```

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.

```

2516 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2517 {
2518   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2519   \tl_gclear:N \g_@@_pre_cell_tl

```

```

2520 \tl_gput_right:Nn \g_@@_array_preamble_tl
2521 {
2522   > {
2523     \dim_set:Nn \l_@@_col_width_dim { #2 }
2524     \@@_cell_begin:
2525     \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2526   }
2527   c
2528   < {
2529     \@@_cell_end_for_w_s:
2530     #1
2531     \@@_adjust_size_box:
2532     \box_use_drop:N \l_@@_cell_box
2533   }
2534 }
2535 \int_gincr:N \c@jCol
2536 \@@_rec_preamble_after_col:n
2537 }

```

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

```

2538 \cs_new_protected:Npn \@@_make_preamble_w_ii:nmmn #1 #2 #3 #4
2539 {
2540   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2541   \tl_gclear:N \g_@@_pre_cell_tl
2542   \tl_gput_right:Nn \g_@@_array_preamble_tl
2543   {
2544     > {

```

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.

```

2545     \dim_set:Nn \l_@@_col_width_dim { #4 }
2546     \hbox_set:Nw \l_@@_cell_box
2547     \@@_cell_begin:
2548     \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2549   }
2550   c
2551   < {
2552     \@@_cell_end:
2553     \hbox_set_end:
2554     #1
2555     \@@_adjust_size_box:
2556     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2557   }
2558 }

```

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

```

2559 \int_gincr:N \c@jCol
2560 \@@_rec_preamble_after_col:n
2561 }

```

```

2562 \cs_new_protected:Npn \@@_special_W:
2563 {
2564   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2565   { \@@_warning:n { W~warning } }
2566 }

```

For S (of siunitx).

```

2567 \cs_new_protected:Npn \@@_S #1 #2
2568 {
2569   \str_if_eq:nnTF { #2 } { [ ] }
2570   { \@@_make_preamble_S:w [ ] }
2571   { \@@_make_preamble_S:w [ ] { #2 } }
2572 }

```

```

2573 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2574 { \@@_make_preamble_S_i:n { #1 } }

2575 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2576 {
2577   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2578   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2579   \tl_gcLEAR:N \g_@@_pre_cell_tl
2580   \tl_gput_right:Nn \g_@@_array_preamble_tl
2581   {
2582     > {

```

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

```

2583       \socket_assign_plug:n { nicematrix / siunitx-wrap } { active }
2584       \keys_set:n { siunitx } { #1 }
2585       \@@_cell_begin:
2586       \siunitx_cell_begin:w
2587     }
2588     c
2589     <
2590     {
2591       \siunitx_cell_end:

```

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

```

2592       \tl_gput_right:Ne \g_@@_cell_after_hook_tl
2593       {
2594         \bool_if:NTF \l__siunitx_table_text_bool
2595         \bool_set_true:N
2596         \bool_set_false:N
2597         \l__siunitx_table_text_bool
2598       }
2599       \@@_cell_end:
2600     }
2601   }

```

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

```

2602     \int_gincr:N \c@jCol
2603     \@@_rec_preamble_after_col:n
2604   }

```

For `(`, `[` and `\{`.

```

2605 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2606 {
2607   \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.

```

2608   \int_if_zero:nTF \c@jCol
2609   {
2610     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2611     {

```

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

```

2612     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2613     \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2614     \@@_rec_preamble:n #2
2615   }
2616   {
2617     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2618         \@@_make_preamble_iv:nn { #1 } { #2 }
2619     }
2620 }
2621 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2622 }
2623 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2624 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2625 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2626 {
2627     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2628     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2629     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2630     {
2631         \@@_error:nn { delimiter~after~opening } { #2 }
2632         \@@_rec_preamble:n
2633     }
2634     { \@@_rec_preamble:n #2 }
2635 }

```

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

```

2636 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2637 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2638 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2639 {
2640     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2641     \tl_if_in:nnTF { ) ] \} } { #2 }
2642     { \@@_make_preamble_v:nnn #1 #2 }
2643     {
2644         \str_if_eq:nnTF { \@@_stop: } { #2 }
2645         {
2646             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2647             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2648             {
2649                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2650                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2651                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2652                 \@@_rec_preamble:n #2
2653             }
2654         }
2655         {
2656             \tl_if_in:nnT { ( [ \{ \left } } { #2 }
2657             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2658             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2659             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2660             \@@_rec_preamble:n #2
2661         }
2662     }
2663 }
2664 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2665 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2666 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2667 {
2668     \str_if_eq:nnTF { \@@_stop: } { #3 }
2669     {
2670         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2671         {
2672             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }

```

```

2673     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2674     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2675     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2676   }
2677   {
2678     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2679     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2680     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2681     \@@_error:nn { double~closing~delimiter } { #2 }
2682   }
2683 }
2684 {
2685   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2686   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2687   \@@_error:nn { double~closing~delimiter } { #2 }
2688   \@@_rec_preamble:n #3
2689 }
2690 }

2691 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2692 { \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 @{...}.

```

2693 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2694 {
2695   \str_if_eq:nnTF { #1 } { < }
2696     \@@_rec_preamble_after_col_i:n
2697     {
2698       \str_if_eq:nnTF { #1 } { @ }
2699         \@@_rec_preamble_after_col_ii:n
2700         {
2701           \str_if_eq:eeTF \l_@@_vlines_clist { all }
2702             {
2703               \tl_gput_right:Nn \g_@@_array_preamble_tl
2704               { ! { \skip_horizontal:N \arrayrulewidth } }
2705             }
2706             {
2707               \clist_if_in:NeT \l_@@_vlines_clist
2708               { \int_eval:n { \c@jCol + 1 } }
2709               {
2710                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2711                 { ! { \skip_horizontal:N \arrayrulewidth } }
2712               }
2713             }
2714           \@@_rec_preamble:n { #1 }
2715         }
2716       }
2717     }

2718 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2719 {
2720   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2721   \@@_rec_preamble_after_col:n
2722 }

```

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.

```

2723 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2724 {
2725   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2726   {

```

```

2727     \tl_gput_right:Nn \g_@@_array_preamble_tl
2728     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2729   }
2730   {
2731     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2732     {
2733       \tl_gput_right:Nn \g_@@_array_preamble_tl
2734       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2735     }
2736     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2737   }
2738   \@@_rec_preamble:n
2739 }

2740 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2741 {
2742   \tl_clear:N \l_tmpa_tl
2743   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2744   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2745 }

```

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.

```

2746 \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.

```

2747 \cs_new_protected:Npn \@@_X #1 #2
2748 {
2749   \str_if_eq:nnTF { #2 } { [ ]
2750     { \@@_make_preamble_X:w [ ] }
2751     { \@@_make_preamble_X:w [ ] #2 }
2752   }
2753   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2754   { \@@_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`).

```

2755 \keys_define:nn { nicematrix / X-column }
2756 { 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.

```

2757 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2758 {

```

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).

```

2759   \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).

```

2760   \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`.

```

2761   \int_zero_new:N \l_@@_weight_int
2762   \int_set_eq:NN \l_@@_weight_int \c_one_int
2763   \@@_keys_p_column:n { #1 }

```


The unknown keys are put in `\l_tmpa_tl`

```

2764 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2765 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2766 {
2767   \@@_error_or_warning:n { negative-weight }
2768   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2769 }
2770 \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).

```

2771 \bool_if:NTF \l_@@_X_columns_aux_bool
2772 {
2773   \@@_make_preamble_ii_iv:nnn
2774   { \l_@@_weight_int \l_@@_X_columns_dim }
2775   { minipage }
2776   { \@@_no_update_width: }
2777 }
2778 {
2779   \tl_gput_right:Nn \g_@@_array_preamble_tl
2780   {
2781     > {
2782       \@@_cell_begin:
2783       \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`.

```

2784 \NotEmpty

```

The following code will nullify the box of the cell.

```

2785 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2786 { \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`.

```

2787 \begin { minipage } { 5 cm } \arraybackslash
2788 }
2789 c
2790 < {
2791   \end { minipage }
2792   \@@_cell_end:
2793 }
2794 }
2795 \int_gincr:N \c@jCol
2796 \@@_rec_preamble_after_col:n
2797 }
2798 }

```

```

2799 \cs_new_protected:Npn \@@_no_update_width:
2800 {
2801   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2802   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2803 }

```

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

```

2804 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2805 {
2806   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2807   { \int_eval:n { \c@jCol + 1 } }
2808   \tl_gput_right:Ne \g_@@_array_preamble_tl
2809   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2810   \@@_rec_preamble:n
2811 }

```

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.

```
2812 \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).

```
2813 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2814 { \@@_fatal:n { Preamble-forgotten } }
2815 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2816 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2817 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2818 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2819 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2820 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox } { @@ _ \token_to_str:N \hline }
```

12 The redefinition of `\multicolumn`

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

```
2821 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2822 {
```

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`.

```
2823 \multispan { #1 }
2824 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2825 \begingroup
2826 \bool_if:NT \c_@@_testphase_table_bool
2827 { \tbl_update_multicolumn_cell_data:n { #1 } }
2828 \cs_set_nopar:Npn \@addamp
2829 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2830 \tl_gclear:N \g_@@_preamble_tl
2831 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2832 \exp_args:No \@mkpream \g_@@_preamble_tl
2833 \addtopreamble \@empty
2834 \endgroup
2835 \bool_if:NT \c_@@_recent_array_bool
2836 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2837 \int_compare:nNnT { #1 } > \c_one_int
2838 {
2839 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2840 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2841 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2842 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2843 {
2844 {
2845 \int_if_zero:nTF \c@jCol
2846 { \int_eval:n { \c@iRow + 1 } }
2847 { \int_use:N \c@iRow }
2848 }
2849 { \int_eval:n { \c@jCol + 1 } }
```

```

2850     {
2851         \int_if_zero:nTF \c@jCol
2852             { \int_eval:n { \c@iRow + 1 } }
2853             { \int_use:N \c@iRow }
2854     }
2855     { \int_eval:n { \c@jCol + #1 } }
2856     { } % for the name of the block
2857 }
2858 }

```

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

```

2859 \RenewDocumentCommand \cellcolor { 0 { } m }
2860 {
2861     \tl_gput_right:Ne \g_@@_pre_code_before_tl
2862     {
2863         \@@_rectanglecolor [ ##1 ]
2864         { \exp_not:n { ##2 } }
2865         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2866         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2867     }
2868     \ignorespaces
2869 }

```

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

```

2870 \cs_set_nopar:Npn \@sharp { #3 }
2871 \@arstrut
2872 \@preamble
2873 \null

```

We add some lines.

```

2874 \int_gadd:Nn \c@jCol { #1 - 1 }
2875 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2876     { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2877 \ignorespaces
2878 }

```

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`.

```

2879 \cs_new_protected:Npn \@_make_m_preamble:n #1
2880 {
2881     \str_case:nnF { #1 }
2882     {
2883         c { \@_make_m_preamble_i:n #1 }
2884         l { \@_make_m_preamble_i:n #1 }
2885         r { \@_make_m_preamble_i:n #1 }
2886         > { \@_make_m_preamble_ii:nn #1 }
2887         ! { \@_make_m_preamble_ii:nn #1 }
2888         @ { \@_make_m_preamble_ii:nn #1 }
2889         | { \@_make_m_preamble_iii:n #1 }
2890         p { \@_make_m_preamble_iv:nnn t #1 }
2891         m { \@_make_m_preamble_iv:nnn c #1 }
2892         b { \@_make_m_preamble_iv:nnn b #1 }
2893         w { \@_make_m_preamble_v:nnnn { } #1 }
2894         W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2895         \q_stop { }
2896     }
2897     {
2898         \cs_if_exist:cTF { NC @ find @ #1 }
2899         {
2900             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2901             \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl

```

```

2902     }
2903     {
2904         \str_if_eq:nnTF { #1 } { S }
2905         { \@@_fatal:n { unknown~column~type~S } }
2906         { \@@_fatal:nn { unknown~column~type } { #1 } }
2907     }
2908 }
2909 }

```

For c, l and r

```

2910 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2911 {
2912     \tl_gput_right:Nn \g_@@_preamble_tl
2913     {
2914         > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2915         #1
2916         < \@@_cell_end:
2917     }

```

We test for the presence of a <.

```

2918     \@@_make_m_preamble_x:n
2919 }

```

For >, ! and @

```

2920 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2921 {
2922     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2923     \@@_make_m_preamble:n
2924 }

```

For |

```

2925 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2926 {
2927     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2928     \@@_make_m_preamble:n
2929 }

```

For p, m and b

```

2930 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2931 {
2932     \tl_gput_right:Nn \g_@@_preamble_tl
2933     {
2934         > {
2935             \@@_cell_begin:
2936             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2937             \mode_leave_vertical:
2938             \arraybackslash
2939             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2940         }
2941         c
2942         < {
2943             \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2944             \end { minipage }
2945             \@@_cell_end:
2946         }
2947     }

```

We test for the presence of a <.

```

2948     \@@_make_m_preamble_x:n
2949 }

```

For w and W

```

2950 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2951 {
2952   \tl_gput_right:Nn \g_@@_preamble_tl
2953   {
2954     > {
2955       \dim_set:Nn \l_@@_col_width_dim { #4 }
2956       \hbox_set:Nw \l_@@_cell_box
2957       \@@_cell_begin:
2958       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2959     }
2960     c
2961     < {
2962       \@@_cell_end:
2963       \hbox_set_end:
2964       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2965       #1
2966       \@@_adjust_size_box:
2967       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2968     }
2969   }

```

We test for the presence of a <.

```

2970   \@@_make_m_preamble_x:n
2971 }

```

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

```

2972 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2973 {
2974   \str_if_eq:nnTF { #1 } { < }
2975   \@@_make_m_preamble_ix:n
2976   { \@@_make_m_preamble:n { #1 } }
2977 }
2978 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2979 {
2980   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2981   \@@_make_m_preamble_x:n
2982 }

```

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`).

```

2983 \cs_new_protected:Npn \@@_put_box_in_flow:
2984 {
2985   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2986   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2987   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2988   { \box_use_drop:N \l_tmpa_box }
2989   \@@_put_box_in_flow_i:
2990 }

```

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

```

2991 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2992 {
2993   \pgfpicture
2994   \@@_qpoint:n { row - 1 }
2995   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2996   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2997   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2998   \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).

```

2999     \tl_if_in:NnTF \l_@@_baseline_tl { line-
3000     {
3001         \int_set:Nn \l_tmpa_int
3002         {
3003             \str_range:Nnn
3004             \l_@@_baseline_tl
3005             6
3006             { \tl_count:o \l_@@_baseline_tl }
3007         }
3008     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3009     }
3010     {
3011     \str_if_eq:eeTF \l_@@_baseline_tl { t }
3012     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3013     {
3014         \str_if_eq:onTF \l_@@_baseline_tl { b }
3015         { \int_set_eq:NN \l_tmpa_int \c_iRow }
3016         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3017     }
3018     \bool_lazy_or:nnT
3019     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3020     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3021     {
3022         \@@_error:n { bad-value-for-baseline }
3023         \int_set_eq:NN \l_tmpa_int \c_one_int
3024     }
3025     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3026     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3027     }
3028     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3029     \endpgfpicture
3030     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3031     \box_use_drop:N \l_tmpa_box
3032     }

```

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).

```

3033     \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3034     {

```

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.

```

3035     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3036     {
3037         \int_compare:nNnT \c_jCol > \c_one_int
3038         {
3039             \box_set_wd:Nn \l_@@_the_array_box
3040             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3041         }
3042     }

```

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

```

3043     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3044     \bool_if:NT \l_@@_caption_above_bool
3045     {

```

```

3046     \tl_if_empty:NF \l_@@_caption_tl
3047     {
3048         \bool_set_false:N \g_@@_caption_finished_bool
3049         \int_gzero:N \c@tabularnote
3050         \@@_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).

```

3051         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3052         {
3053             \tl_gput_right:Ne \g_@@_aux_tl
3054             {
3055                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3056                 { \int_use:N \g_@@_notes_caption_int }
3057             }
3058             \int_gzero:N \g_@@_notes_caption_int
3059         }
3060     }
3061 }

```

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

```

3062     \hbox
3063     {
3064         \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.

```

3065         \@@_create_extra_nodes:
3066         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3067     }

```

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

```

3068     \bool_lazy_any:nT
3069     {
3070         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3071         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3072         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3073     }
3074     \@@_insert_tabularnotes:
3075     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3076     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3077     \end { minipage }
3078 }

```

```

3079 \cs_new_protected:Npn \@@_insert_caption:
3080 {
3081     \tl_if_empty:NF \l_@@_caption_tl
3082     {
3083         \cs_if_exist:NTF \@capttype
3084         { \@@_insert_caption_i: }
3085         { \@@_error:n { caption~outside~float } }
3086     }
3087 }

```

```

3088 \cs_new_protected:Npn \@@_insert_caption_i:
3089 {
3090     \group_begin:

```

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

```
3091 \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.

```
3092 \IfPackageLoadedT { floatrow }
3093 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3094 \tl_if_empty:NTF \l_@@_short_caption_tl
3095 { \caption }
3096 { \caption [ \l_@@_short_caption_tl ] }
3097 { \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.

```
3098 \bool_if:NF \g_@@_caption_finished_bool
3099 {
3100   \bool_gset_true:N \g_@@_caption_finished_bool
3101   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3102   \int_gzero:N \c@tabularnote
3103 }
3104 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3105 \group_end:
3106 }

3107 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3108 {
3109   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3110   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3111 }

3112 \cs_new_protected:Npn \@@_insert_tabularnotes:
3113 {
3114   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3115   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3116   \skip_vertical:N 0.65ex
```

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

```
3117 \group_begin:
3118 \l_@@_notes_code_before_tl
3119 \tl_if_empty:NF \g_@@_tabularnote_tl
3120 {
3121   \g_@@_tabularnote_tl \par
3122   \tl_gclear:N \g_@@_tabularnote_tl
3123 }
```

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.

```
3124 \int_compare:nNnT \c@tabularnote > \c_zero_int
3125 {
3126   \bool_if:NTF \l_@@_notes_para_bool
3127   {
3128     \begin { tabularnotes* }
3129     \seq_map_inline:Nn \g_@@_notes_seq
3130     { \@@_one_tabularnote:nn ##1 }
3131     \strut
3132     \end { tabularnotes* }
3133 }
```

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

```
3133 \par
```



```

3134     }
3135     {
3136       \tabularnotes
3137       \seq_map_inline:Nn \g_@@_notes_seq
3138       { \@@_one_tabularnote:nn #1 }
3139       \strut
3140       \endtabularnotes
3141     }
3142   }
3143   \unskip
3144   \group_end:
3145   \bool_if:NT \l_@@_notes_bottomrule_bool
3146   {
3147     \IfPackageLoadedTF { booktabs }
3148     {

```

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

```

3149       \skip_vertical:N \aboverulesep

```

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

```

3150       { \CT@arc@ \hrule height \heavyrulewidth }
3151     }
3152     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3153   }
3154   \l_@@_notes_code_after_tl
3155   \seq_gclear:N \g_@@_notes_seq
3156   \seq_gclear:N \g_@@_notes_in_caption_seq
3157   \int_gzero:N \c@tabularnote
3158 }

```

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 currying.

```

3159 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3160 {
3161   \tl_if_novalue:nTF { #1 }
3162     { \item }
3163     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3164 }

```

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`.

```

3165 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3166 {
3167   \pgfpicture
3168     \@@_qpoint:n { row - 1 }
3169     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3170     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3171     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3172   \endpgfpicture
3173   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3174   \int_if_zero:nT \l_@@_first_row_int
3175     {
3176       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3177       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3178     }
3179   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3180 }

```

Now, the general case.

```

3181 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3182 {

```

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

```
3183 \str_if_eq:eeT \l_@@_baseline_tl { t }
3184 { \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`.

```
3185 \pgfpicture
3186 \@@_qpoint:n { row - 1 }
3187 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3188 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3189 {
3190   \int_set:Nn \l_tmpa_int
3191   {
3192     \str_range:Nnn
3193     \l_@@_baseline_tl
3194     6
3195     { \tl_count:o \l_@@_baseline_tl }
3196   }
3197   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3198 }
3199 {
3200   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3201   \bool_lazy_or:nnT
3202   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3203   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3204   {
3205     \@@_error:n { bad-value-for~baseline }
3206     \int_set:Nn \l_tmpa_int 1
3207   }
3208   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3209 }
3210 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3211 \endpgfpicture
3212 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3213 \int_if_zero:nT \l_@@_first_row_int
3214 {
3215   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3216   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3217 }
3218 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3219 }
```

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.

```
3220 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3221 {
```

We will compute the real width of both delimiters used.

```
3222 \dim_zero_new:N \l_@@_real_left_delim_dim
3223 \dim_zero_new:N \l_@@_real_right_delim_dim
3224 \hbox_set:Nn \l_tmpb_box
3225 {
3226   \m@th % added 2024/11/21
3227   \c_math_toggle_token
3228   \left #1
3229   \vcenter
3230   {
3231     \vbox_to_ht:nn
3232     { \box_ht_plus_dp:N \l_tmpa_box }
3233     { }
3234   }
3235   \right .
```

```

3236     \c_math_toggle_token
3237   }
3238   \dim_set:Nn \l_@@_real_left_delim_dim
3239     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3240   \hbox_set:Nn \l_tmpb_box
3241     {
3242     \m@th % added 2024/11/21
3243     \c_math_toggle_token
3244     \left .
3245     \vbox_to_ht:nn
3246       { \box_ht_plus_dp:N \l_tmpa_box }
3247       { }
3248     \right #2
3249     \c_math_toggle_token
3250   }
3251   \dim_set:Nn \l_@@_real_right_delim_dim
3252     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3253   \skip_horizontal:N \l_@@_left_delim_dim
3254   \skip_horizontal:N -\l_@@_real_left_delim_dim
3255   \@@_put_box_in_flow:
3256   \skip_horizontal:N \l_@@_right_delim_dim
3257   \skip_horizontal:N -\l_@@_real_right_delim_dim
3258 }

```

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).

```

3259 \NewDocumentEnvironment { @@-normal-syntax } { }

```

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

```

3260 {
3261   \peek_remove_spaces:n
3262   {
3263     \peek_meaning:NTF \end
3264     \@@_analyze_end:Nn
3265     {
3266       \@@_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`).

```

3267     \@@_array:o \g_@@_array_preamble_tl
3268   }
3269 }
3270 }
3271 {
3272   \@@_create_col_nodes:
3273   \endarray
3274 }

```

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

```

3275 \NewDocumentEnvironment { @@-light-syntax } { b }
3276 {

```

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.

```

3277 \tl_if_empty:nT { #1 }
3278   { \@@_fatal:n { empty~environment } }
3279 \tl_if_in:nnT { #1 } { & }
3280   { \@@_fatal:n { ampersand~in~light~syntax } }
3281 \tl_if_in:nnT { #1 } { \ }
3282   { \@@_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`.

```

3283 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3284 }

```

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.

```

3285 {
3286 \@@_create_col_nodes:
3287 \endarray
3288 }
3289 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3290 {
3291 \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).

```

3292 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3293 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3294 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3295   \seq_set_split:Nee
3296   \seq_set_split:Nnon
3297   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3298 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3299 \tl_if_empty:NF \l_tmpa_tl
3300   { \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.

```

3301 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3302   { \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`).

```

3303 \tl_build_begin:N \l_@@_new_body_tl
3304 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3305 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3306 \@@_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).

```

3307   \seq_map_inline:Nn \l_@@_rows_seq
3308   {
3309     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3310     \@@_line_with_light_syntax:n { #1 }
3311   }
3312   \tl_build_end:N \l_@@_new_body_tl
3313   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3314   {
3315     \int_set:Nn \l_@@_last_col_int
3316     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3317   }

```

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.

```

3318   \@@_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`).

```

3319   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3320   }
3321   \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3322   \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3323   {
3324     \seq_clear_new:N \l_@@_cells_seq
3325     \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3326     \int_set:Nn \l_@@_nb_cols_int
3327     {
3328       \int_max:nn
3329       \l_@@_nb_cols_int
3330       { \seq_count:N \l_@@_cells_seq }
3331     }
3332     \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3333     \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3334     \seq_map_inline:Nn \l_@@_cells_seq
3335     { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3336   }

```

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`.

```

3337   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3338   {
3339     \str_if_eq:eeT \g_@@_name_env_str { #2 }
3340     { \@@_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.

```

3341     \end { #2 }
3342   }

```

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`).

```

3343   \cs_new:Npn \@@_create_col_nodes:
3344   {
3345     \crrc
3346     \int_if_zero:nT \l_@@_first_col_int
3347     {
3348       \omit

```

```

3349     \hbox_overlap_left:n
3350     {
3351         \bool_if:NT \l_@@_code_before_bool
3352         { \pgfsys@markposition { \@@_env: - col - 0 } }
3353         \pgfpicture
3354         \pgfrememberpicturepositiononpagetrue
3355         \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3356         \str_if_empty:NF \l_@@_name_str
3357         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3358         \endpgfpicture
3359         \skip_horizontal:N 2\col@sep
3360         \skip_horizontal:N \g_@@_width_first_col_dim
3361     }
3362     &
3363 }
3364 \omit

```

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

```

3365     \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`).

```

3366     \int_if_zero:nTF \l_@@_first_col_int
3367     {
3368         \bool_if:NT \l_@@_code_before_bool
3369         {
3370             \hbox
3371             {
3372                 \skip_horizontal:N -0.5\arrayrulewidth
3373                 \pgfsys@markposition { \@@_env: - col - 1 }
3374                 \skip_horizontal:N 0.5\arrayrulewidth
3375             }
3376         }
3377         \pgfpicture
3378         \pgfrememberpicturepositiononpagetrue
3379         \pgfcoordinate { \@@_env: - col - 1 }
3380         { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3381         \str_if_empty:NF \l_@@_name_str
3382         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3383         \endpgfpicture
3384     }
3385     {
3386         \bool_if:NT \l_@@_code_before_bool
3387         {
3388             \hbox
3389             {
3390                 \skip_horizontal:N 0.5\arrayrulewidth
3391                 \pgfsys@markposition { \@@_env: - col - 1 }
3392                 \skip_horizontal:N -0.5\arrayrulewidth
3393             }
3394         }
3395         \pgfpicture
3396         \pgfrememberpicturepositiononpagetrue
3397         \pgfcoordinate { \@@_env: - col - 1 }
3398         { \pgfpint { 0.5 \arrayrulewidth } \c_zero_dim }
3399         \str_if_empty:NF \l_@@_name_str
3400         { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3401         \endpgfpicture
3402     }

```

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.

```

3403 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3404 \bool_if:NF \l_@@_auto_columns_width_bool
3405   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3406   {
3407     \bool_lazy_and:nnTF
3408       \l_@@_auto_columns_width_bool
3409       { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3410       { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3411       { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3412     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3413   }
3414 \skip_horizontal:N \g_tmpa_skip
3415 \hbox
3416   {
3417     \bool_if:NT \l_@@_code_before_bool
3418     {
3419       \hbox
3420         {
3421           \skip_horizontal:N -0.5\arrayrulewidth
3422           \pgfsys@markposition { \@@_env: - col - 2 }
3423           \skip_horizontal:N 0.5\arrayrulewidth
3424         }
3425       }
3426     \pgfpicture
3427     \pgfrememberpicturepositiononpagetrue
3428     \pgfcoordinate { \@@_env: - col - 2 }
3429     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3430     \str_if_empty:NF \l_@@_name_str
3431     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3432     \endpgfpicture
3433   }

```

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.

```

3434 \int_gset_eq:NN \g_tmpa_int \c_one_int
3435 \bool_if:NTF \g_@@_last_col_found_bool
3436   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3437   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3438   {
3439     &
3440     \omit
3441     \int_gincr:N \g_tmpa_int

```

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

```

3442     \skip_horizontal:N \g_tmpa_skip
3443     \bool_if:NT \l_@@_code_before_bool
3444     {
3445       \hbox
3446         {
3447           \skip_horizontal:N -0.5\arrayrulewidth
3448           \pgfsys@markposition
3449             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3450           \skip_horizontal:N 0.5\arrayrulewidth
3451         }
3452     }

```

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

```

3453     \pgfpicture
3454     \pgfrememberpicturepositiononpagetrue
3455     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3456     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3457     \str_if_empty:NF \l_@@_name_str

```

```

3458     {
3459         \pgfnodealias
3460         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3461         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3462     }
3463 \endpgfpicture
3464 }

3465 &
3466 \omit

```

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

```

3467 \int_if_zero:nT \g_@@_col_total_int
3468   { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3469 \skip_horizontal:N \g_tmpa_skip
3470 \int_gincr:N \g_tmpa_int
3471 \bool_lazy_any:nF
3472   {
3473     \g_@@_delims_bool
3474     \l_@@_tabular_bool
3475     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3476     \l_@@_exterior_arraycolsep_bool
3477     \l_@@_bar_at_end_of_pream_bool
3478   }
3479   { \skip_horizontal:N -\col@sep }
3480 \bool_if:NT \l_@@_code_before_bool
3481   {
3482     \hbox
3483     {
3484       \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.

```

3485     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3486     { \skip_horizontal:N -\arraycolsep }
3487 \pgfsys@markposition
3488   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489 \skip_horizontal:N 0.5\arrayrulewidth
3490 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3491   { \skip_horizontal:N \arraycolsep }
3492   }
3493 }
3494 \pgfpicture
3495 \pgfrememberpicturepositiononpagetrue
3496 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3497   {
3498     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3499     {
3500       \pgfpoint
3501       { - 0.5 \arrayrulewidth - \arraycolsep }
3502       \c_zero_dim
3503     }
3504     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3505   }
3506 \str_if_empty:NF \l_@@_name_str
3507   {
3508     \pgfnodealias
3509     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3510     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3511   }
3512 \endpgfpicture

```



```

3513 \bool_if:NT \g_@@_last_col_found_bool
3514 {
3515   \hbox_overlap_right:n
3516   {
3517     \skip_horizontal:N \g_@@_width_last_col_dim
3518     \skip_horizontal:N \col@sep
3519     \bool_if:NT \l_@@_code_before_bool
3520     {
3521       \pgfsys@markposition
3522       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3523     }
3524     \pgfpicture
3525     \pgfrememberpicturepositiononpagetrue
3526     \pgfcoordinate
3527     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3528     \pgfpointorigin
3529     \str_if_empty:NF \l_@@_name_str
3530     {
3531       \pgfnodealias
3532       {
3533         \l_@@_name_str - col
3534         - \int_eval:n { \g_@@_col_total_int + 1 }
3535       }
3536       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3537     }
3538     \endpgfpicture
3539   }
3540 }
3541 % \cr
3542 }

```

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

```

3543 \tl_const:Nn \c_@@_preamble_first_col_tl
3544 {
3545   >
3546   {

```

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:
3548     \bool_gset_true:N \g_@@_after_col_zero_bool
3549     \@@_begin_of_row:
3550     \hbox_set:Nw \l_@@_cell_box
3551     \@@_math_toggle:
3552     \@@_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”.

```

3553     \int_compare:nNnT \c@iRow > \c_zero_int
3554     {
3555       \bool_lazy_or:nnT
3556       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3557       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3558       {
3559         \l_@@_code_for_first_col_tl
3560         \xglobal \colorlet { nicematrix-first-col } { . }
3561       }
3562     }
3563   }

```

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`.

```

3564   l

```

```

3565 <
3566 {
3567   \@@_math_toggle:
3568   \hbox_set_end:
3569   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3570   \@@_adjust_size_box:
3571   \@@_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.

```

3572   \dim_gset:Nn \g_@@_width_first_col_dim
3573   { \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.

```

3574   \hbox_overlap_left:n
3575   {
3576     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3577     \@@_node_for_cell:
3578     { \box_use_drop:N \l_@@_cell_box }
3579     \skip_horizontal:N \l_@@_left_delim_dim
3580     \skip_horizontal:N \l_@@_left_margin_dim
3581     \skip_horizontal:N \l_@@_extra_left_margin_dim
3582   }
3583   \bool_gset_false:N \g_@@_empty_cell_bool
3584   \skip_horizontal:N -2\col@sep
3585 }
3586 }

```

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

```

3587 \tl_const:Nn \c_@@_preamble_last_col_tl
3588 {
3589   >
3590   {
3591     \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).

```

3592     \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.

```

3593     \bool_gset_true:N \g_@@_last_col_found_bool
3594     \int_gincr:N \c@jCol
3595     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3596     \hbox_set:Nw \l_@@_cell_box
3597     \@@_math_toggle:
3598     \@@_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”.

```

3599     \int_compare:nNnT \c@iRow > \c_zero_int
3600     {
3601       \bool_lazy_or:nnT
3602       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3603       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3604       {
3605         \l_@@_code_for_last_col_tl
3606         \xglobal \colorlet { nicematrix-last-col } { . }
3607       }
3608     }
3609   }
3610   l
3611   <
3612   {
3613     \@@_math_toggle:
3614     \hbox_set_end:
3615     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

```

3616 \@@_adjust_size_box:
3617 \@@_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.

```

3618 \dim_gset:Nn \g_@@_width_last_col_dim
3619 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3620 \skip_horizontal:N -2\col@sep

```

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

```

3621 \hbox_overlap_right:n
3622 {
3623   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3624   {
3625     \skip_horizontal:N \l_@@_right_delim_dim
3626     \skip_horizontal:N \l_@@_right_margin_dim
3627     \skip_horizontal:N \l_@@_extra_right_margin_dim
3628     \@@_node_for_cell:
3629   }
3630 }
3631 \bool_gset_false:N \g_@@_empty_cell_bool
3632 }
3633 }

```

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

```

3634 \NewDocumentEnvironment { NiceArray } { }
3635 {
3636   \bool_gset_false:N \g_@@_delims_bool
3637   \str_if_empty:NT \g_@@_name_env_str
3638   { \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).

```

3639   \NiceArrayWithDelims . .
3640 }
3641 { \endNiceArrayWithDelims }

```

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

```

3642 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3643 {
3644   \NewDocumentEnvironment { #1 NiceArray } { }
3645   {
3646     \bool_gset_true:N \g_@@_delims_bool
3647     \str_if_empty:NT \g_@@_name_env_str
3648     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3649     \@@_test_if_math_mode:
3650     \NiceArrayWithDelims #2 #3
3651   }
3652   { \endNiceArrayWithDelims }
3653 }
3654 \@@_def_env:nnn p ( )
3655 \@@_def_env:nnn b [ ]
3656 \@@_def_env:nnn B \{ \}
3657 \@@_def_env:nnn v | |
3658 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3659 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3660 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3661 {
3662   \bool_set_false:N \l_@@_preamble_bool
3663   \tl_clear:N \l_tmpa_tl
3664   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3665     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3666   \tl_put_right:Nn \l_tmpa_tl
3667     {
3668       *
3669       {
3670         \int_case:nnF \l_@@_last_col_int
3671         {
3672           { -2 } { \c@MaxMatrixCols }
3673           { -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).

```

3674         }
3675         { \int_eval:n { \l_@@_last_col_int - 1 } }
3676       }
3677     { #2 }
3678   }
3679   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3680   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3681 }
3682 \clist_map_inline:nn { p , b , B , v , V }
3683 {
3684   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3685   {
3686     \bool_gset_true:N \g_@@_delims_bool
3687     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3688     \int_if_zero:nT \l_@@_last_col_int
3689     {
3690       \bool_set_true:N \l_@@_last_col_without_value_bool
3691       \int_set:Nn \l_@@_last_col_int { -1 }
3692     }
3693     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3694     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3695   }
3696   { \use:c { end #1 NiceArray } }
3697 }

```

We define also an environment `{NiceMatrix}`

```

3698 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3699 {
3700   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3701   \int_if_zero:nT \l_@@_last_col_int
3702   {
3703     \bool_set_true:N \l_@@_last_col_without_value_bool
3704     \int_set:Nn \l_@@_last_col_int { -1 }
3705   }
3706   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3707   \bool_lazy_or:nnT
3708     { \clist_if_empty_p:N \l_@@_vlines_clist }
3709     { \l_@@_except_borders_bool }
3710     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3711   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3712 }
3713 { \endNiceArray }

```

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

```
3714 \cs_new_protected:Npn \@@_NotEmpty:
3715 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3716 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3717 {
```

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`.

```
3718   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3719     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3720   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3721   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3722   \tl_if_empty:NF \l_@@_short_caption_tl
3723     {
3724     \tl_if_empty:NT \l_@@_caption_tl
3725       {
3726         \@@_error_or_warning:n { short-caption-without-caption }
3727         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3728       }
3729     }
3730   \tl_if_empty:NF \l_@@_label_tl
3731     {
3732     \tl_if_empty:NT \l_@@_caption_tl
3733       { \@@_error_or_warning:n { label-without-caption } }
3734     }
3735   \NewDocumentEnvironment { TabularNote } { b }
3736     {
3737     \bool_if:NTF \l_@@_in_code_after_bool
3738       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3739       {
3740         \tl_if_empty:NF \g_@@_tabularnote_tl
3741           { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3742         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3743       }
3744     }
3745     { }
3746   \@@_settings_for_tabular:
3747   \NiceArray { #2 }
3748 }
3749 { \endNiceArray }
3750 \cs_new_protected:Npn \@@_settings_for_tabular:
3751 {
3752   \bool_set_true:N \l_@@_tabular_bool
3753   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3754   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3755   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3756 }
```

```
3757 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3758 {
3759   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3760   \dim_zero_new:N \l_@@_width_dim
3761   \dim_set:Nn \l_@@_width_dim { #1 }
3762   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3763   \@@_settings_for_tabular:
3764   \NiceArray { #3 }
3765 }
3766 {
3767   \endNiceArray
```

```

3768 \int_if_zero:nT \g_@@_total_X_weight_int
3769 { \@@_error:n { NiceTabularX~without~X } }
3770 }

3771 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3772 {
3773 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3774 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3775 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3776 \@@_settings_for_tabular:
3777 \NiceArray { #3 }
3778 }
3779 { \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`).

```

3780 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3781 {
3782 \bool_lazy_all:nT
3783 {
3784 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3785 \l_@@_hvlines_bool
3786 { ! \g_@@_delims_bool }
3787 { ! \l_@@_except_borders_bool }
3788 }
3789 {
3790 \bool_set_true:N \l_@@_except_borders_bool
3791 \clist_if_empty:NF \l_@@_corners_clist
3792 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3793 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3794 {
3795 \@@_stroke_block:nnn
3796 {
3797 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3798 draw = \l_@@_rules_color_tl
3799 }
3800 { 1-1 }
3801 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3802 }
3803 }
3804 }

```

```

3805 \cs_new_protected:Npn \@@_after_array:
3806 {

```

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`.

```

3807 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3808 \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.

```
3809   \bool_if:NT \g_@@_last_col_found_bool
3810     { \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`.

```
3811   \bool_if:NT \l_@@_last_col_without_value_bool
3812     { \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.

```
3813   \bool_if:NT \l_@@_last_row_without_value_bool
3814     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3815   \tl_gput_right:Ne \g_@@_aux_tl
3816     {
3817       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3818         {
3819           \int_use:N \l_@@_first_row_int ,
3820           \int_use:N \c@iRow ,
3821           \int_use:N \g_@@_row_total_int ,
3822           \int_use:N \l_@@_first_col_int ,
3823           \int_use:N \c@jCol ,
3824           \int_use:N \g_@@_col_total_int
3825         }
3826     }
```

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`).

```
3827   \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3828     {
3829       \tl_gput_right:Ne \g_@@_aux_tl
3830         {
3831           \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3832             { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3833         }
3834     }
3835   \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3836     {
3837       \tl_gput_right:Ne \g_@@_aux_tl
3838         {
3839           \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3840             { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3841           \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3842             { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3843         }
3844     }
```

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

```
3845   \@@_create_diag_nodes:
```

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

```
3846   \pgfpicture
3847   \int_step_inline:nn \c@iRow
3848     {
3849       \pgfnodealias
3850         { \@@_env: - ##1 - last }
3851         { \@@_env: - ##1 - \int_use:N \c@jCol }
3852     }
3853   \int_step_inline:nn \c@jCol
3854     {
3855       \pgfnodealias
```

```

3856     { \@@_env: - last - ##1 }
3857     { \@@_env: - \int_use:N \c@iRow - ##1 }
3858   }
3859   \str_if_empty:NF \l_@@_name_str
3860   {
3861     \int_step_inline:nn \c@iRow
3862     {
3863       \pgfnodealias
3864       { \l_@@_name_str - ##1 - last }
3865       { \@@_env: - ##1 - \int_use:N \c@jCol }
3866     }
3867     \int_step_inline:nn \c@jCol
3868     {
3869       \pgfnodealias
3870       { \l_@@_name_str - last - ##1 }
3871       { \@@_env: - \int_use:N \c@iRow - ##1 }
3872     }
3873   }
3874   \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.

```

3875   \bool_if:NT \l_@@_parallelize_diags_bool
3876   {
3877     \int_gzero_new:N \g_@@_ddots_int
3878     \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.

```

3879     \dim_gzero_new:N \g_@@_delta_x_one_dim
3880     \dim_gzero_new:N \g_@@_delta_y_one_dim
3881     \dim_gzero_new:N \g_@@_delta_x_two_dim
3882     \dim_gzero_new:N \g_@@_delta_y_two_dim
3883   }
3884   \int_zero_new:N \l_@@_initial_i_int
3885   \int_zero_new:N \l_@@_initial_j_int
3886   \int_zero_new:N \l_@@_final_i_int
3887   \int_zero_new:N \l_@@_final_j_int
3888   \bool_set_false:N \l_@@_initial_open_bool
3889   \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.

```

3890   \bool_if:NT \l_@@_small_bool
3891   {
3892     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3893     \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.

```

3894     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3895     { 0.6 \l_@@_xdots_shorten_start_dim }
3896     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3897     { 0.6 \l_@@_xdots_shorten_end_dim }
3898   }

```

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.).

```
3899 \@@_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.

```
3900 \clist_if_empty:NF \l_@@_corners_clist
3901 {
3902   \bool_if:NTF \l_@@_no_cell_nodes_bool
3903     { \@@_error:n { corners~with~no~cell~nodes } }
3904     { \@@_compute_corners: }
3905 }
```

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

```
3906 \@@_adjust_pos_of_blocks_seq:
3907 \@@_deal_with_rounded_corners:
3908 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3909 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3910 \IfPackageLoadedT { tikz }
3911 {
3912   \tikzset
3913   {
3914     every-picture / .style =
3915     {
3916       overlay ,
3917       remember~picture ,
3918       name~prefix = \@@_env: -
3919     }
3920   }
3921 }
3922 \bool_if:NT \c_@@_recent_array_bool
3923 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3924 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3925 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3926 \cs_set_eq:NN \OverBrace \@@_OverBrace
3927 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3928 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3929 \cs_set_eq:NN \line \@@_line
3930 \g_@@_pre_code_after_tl
3931 \tl_gclear:N \g_@@_pre_code_after_tl
```

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

```
3932 \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).

```
3933 \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`).

```
3934 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3935 { \@@_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:`.

```

3936     \bool_set_true:N \l_@@_in_code_after_bool
3937     \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3938     \scan_stop:
3939     \tl_gclear:N \g_nicematrix_code_after_tl
3940     \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.

```

3941     \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3942     \tl_if_empty:NF \g_@@_pre_code_before_tl
3943     {
3944         \tl_gput_right:Ne \g_@@_aux_tl
3945         {
3946             \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3947             { \exp_not:o \g_@@_pre_code_before_tl }
3948         }
3949         \tl_gclear:N \g_@@_pre_code_before_tl
3950     }
3951     \tl_if_empty:NF \g_nicematrix_code_before_tl
3952     {
3953         \tl_gput_right:Ne \g_@@_aux_tl
3954         {
3955             \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3956             { \exp_not:o \g_nicematrix_code_before_tl }
3957         }
3958         \tl_gclear:N \g_nicematrix_code_before_tl
3959     }

3960     \str_gclear:N \g_@@_name_env_str
3961     \@@_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`.

```

3962     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3963 }

```

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`.

```

3964 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3965 { \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).

```

3966 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3967 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3968 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3969 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3970 }

```

The following command must *not* be protected.

```

3971 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3972 {
3973   { #1 }
3974   { #2 }
3975   {
3976     \int_compare:nNnTF { #3 } > { 98 }
3977     { \int_use:N \c@iRow }
3978     { #3 }
3979   }
3980   {
3981     \int_compare:nNnTF { #4 } > { 98 }
3982     { \int_use:N \c@jCol }
3983     { #4 }
3984   }
3985   { #5 }
3986 }

```

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).

```

3987 \hook_gput_code:nnn { begindocument } { . }
3988 {
3989   \cs_new_protected:Npe \@@_draw_dotted_lines:
3990   {
3991     \c_@@_pgfortikzpicture_tl
3992     \@@_draw_dotted_lines_i:
3993     \c_@@_endpgfortikzpicture_tl
3994   }
3995 }

```

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

```

3996 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3997 {
3998   \pgfrememberpicturepositiononpagetrue
3999   \pgf@relevantforpicturesizefalse
4000   \g_@@_HVdotsfor_lines_tl
4001   \g_@@_Vdots_lines_tl
4002   \g_@@_Ddots_lines_tl
4003   \g_@@_Iddots_lines_tl
4004   \g_@@_Cdots_lines_tl
4005   \g_@@_Ldots_lines_tl
4006 }

4007 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4008 {
4009   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4010   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4011 }

```

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

```

4012 \pgfdeclareshape { @@_diag_node }
4013 {
4014   \savedanchor { \five }
4015   {
4016     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

4017     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4018   }
4019   \anchor { 5 } { \five }
4020   \anchor { center } { \pgfpointorigin }
4021   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4022   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4023   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4024   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4025   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4026   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4027   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4028   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4029   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4030   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4031 }

```

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).

```

4032 \cs_new_protected:Npn \@@_create_diag_nodes:
4033 {
4034   \pgfpicture
4035   \pgfrememberpicturepositiononpagetrue
4036   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4037   {
4038     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4039     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4040     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4041     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4042     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4043     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4044     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4045     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4046     \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.

```

4047     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4048     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4049     \pgfnode { \@@_diag_node } { center } { } { \@@_env: - ##1 } { }
4050     \str_if_empty:NF \l_@@_name_str
4051     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4052   }

```

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

```

4053   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4054   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4055   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4056   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4057   \pgfcoordinate
4058   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4059   \pgfnodealias
4060   { \@@_env: - last }
4061   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4062   \str_if_empty:NF \l_@@_name_str
4063   {
4064     \pgfnodealias
4065     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4066     { \@@_env: - \int_use:N \l_tmpa_int }
4067     \pgfnodealias
4068     { \l_@@_name_str - last }
4069     { \@@_env: - last }
4070   }

```


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.

```

4086     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4087     \if_int_compare:w #3 = \c_one_int
4088     \bool_set_true:N \l_@@_final_open_bool
4089     \else:
4090     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4091     \bool_set_true:N \l_@@_final_open_bool
4092     \fi:
4093     \fi:
4094     \else:
4095     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4096     \if_int_compare:w #4 = -1
4097     \bool_set_true:N \l_@@_final_open_bool
4098     \fi:
4099     \else:
4100     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4101     \if_int_compare:w #4 = \c_one_int
4102     \bool_set_true:N \l_@@_final_open_bool
4103     \fi:
4104     \fi:
4105     \fi:
4106     \fi:
4107     \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.

```

4108     {

```

We do a step backwards.

```

4109         \int_sub:Nn \l_@@_final_i_int { #3 }
4110         \int_sub:Nn \l_@@_final_j_int { #4 }
4111         \bool_set_true:N \l_@@_stop_loop_bool
4112     }

```

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`.

```

4113     {
4114     \cs_if_exist:cTF
4115     {
4116     @@ _ dotted _
4117     \int_use:N \l_@@_final_i_int -
4118     \int_use:N \l_@@_final_j_int
4119     }
4120     {
4121     \int_sub:Nn \l_@@_final_i_int { #3 }
4122     \int_sub:Nn \l_@@_final_j_int { #4 }
4123     \bool_set_true:N \l_@@_final_open_bool
4124     \bool_set_true:N \l_@@_stop_loop_bool
4125     }
4126     {
4127     \cs_if_exist:cTF
4128     {
4129     pgf @ sh @ ns @ \@@_env:
4130     - \int_use:N \l_@@_final_i_int
4131     - \int_use:N \l_@@_final_j_int
4132     }
4133     { \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.

```

4134     {

```

```

4135         \cs_set_nopar:cpn
4136         {
4137             @@ _ dotted _
4138             \int_use:N \l_@@_final_i_int -
4139             \int_use:N \l_@@_final_j_int
4140         }
4141         { }
4142     }
4143 }
4144 }
4145 }

```

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

```

4146     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4147     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4148     \bool_do_until:Nn \l_@@_stop_loop_bool
4149     {
4150         \int_sub:Nn \l_@@_initial_i_int { #3 }
4151         \int_sub:Nn \l_@@_initial_j_int { #4 }
4152         \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.

```

4153         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4154         \if_int_compare:w #3 = \c_one_int
4155             \bool_set_true:N \l_@@_initial_open_bool
4156         \else:

```

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

```

4157         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4158             \bool_set_true:N \l_@@_initial_open_bool
4159         \fi:
4160     \fi:
4161 \else:
4162     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4163         \if_int_compare:w #4 = \c_one_int
4164             \bool_set_true:N \l_@@_initial_open_bool
4165         \fi:
4166     \else:
4167         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4168             \if_int_compare:w #4 = -1
4169                 \bool_set_true:N \l_@@_initial_open_bool
4170             \fi:
4171         \fi:
4172     \fi:
4173 \fi:
4174 \bool_if:NTF \l_@@_initial_open_bool
4175 {
4176     \int_add:Nn \l_@@_initial_i_int { #3 }
4177     \int_add:Nn \l_@@_initial_j_int { #4 }
4178     \bool_set_true:N \l_@@_stop_loop_bool
4179 }
4180 {
4181     \cs_if_exist:cTF
4182     {
4183         @@ _ dotted _
4184         \int_use:N \l_@@_initial_i_int -
4185         \int_use:N \l_@@_initial_j_int
4186     }

```

```

4187     {
4188         \int_add:Nn \l_@@_initial_i_int { #3 }
4189         \int_add:Nn \l_@@_initial_j_int { #4 }
4190         \bool_set_true:N \l_@@_initial_open_bool
4191         \bool_set_true:N \l_@@_stop_loop_bool
4192     }
4193     {
4194         \cs_if_exist:cTF
4195         {
4196             pgf @ sh @ ns @ \@@_env:
4197             - \int_use:N \l_@@_initial_i_int
4198             - \int_use:N \l_@@_initial_j_int
4199         }
4200         { \bool_set_true:N \l_@@_stop_loop_bool }
4201         {
4202             \cs_set_nopar:cpn
4203             {
4204                 @@ _ dotted _
4205                 \int_use:N \l_@@_initial_i_int -
4206                 \int_use:N \l_@@_initial_j_int
4207             }
4208             { }
4209         }
4210     }
4211 }
4212 }

```

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.

```

4213 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4214 {
4215     { \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`.

```

4216     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4217     { \int_use:N \l_@@_final_i_int }
4218     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4219     { } % for the name of the block
4220 }
4221 }

```

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.

```

4222 \cs_new_protected:Npn \@@_open_shorten:
4223 {
4224     \bool_if:NT \l_@@_initial_open_bool
4225     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4226     \bool_if:NT \l_@@_final_open_bool
4227     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4228 }

```

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).

```

4229 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4230 {
4231     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```



```

4232 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4233 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4234 \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`.

```

4235 \seq_if_empty:NF \g_@@_submatrix_seq
4236 {
4237   \seq_map_inline:Nn \g_@@_submatrix_seq
4238     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4239 }
4240 }

```

`#1` and `#2` are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. `#3`, `#4`, `#5` and `#6` are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

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

```

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

```

4241 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4242 {
4243   \if_int_compare:w #3 > #1
4244   \else:
4245     \if_int_compare:w #1 > #5
4246     \else:
4247       \if_int_compare:w #4 > #2
4248       \else:
4249         \if_int_compare:w #2 > #6
4250         \else:
4251           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4252           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4253           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4254           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4255         \fi:
4256       \fi:
4257     \fi:
4258   \fi:
4259 }

4260 \cs_new_protected:Npn \@@_set_initial_coords:
4261 {
4262   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4263   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4264 }
4265 \cs_new_protected:Npn \@@_set_final_coords:
4266 {

```

```

4267 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4268 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4269 }
4270 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4271 {
4272 \pgfpointanchor
4273 {
4274 \@@_env:
4275 - \int_use:N \l_@@_initial_i_int
4276 - \int_use:N \l_@@_initial_j_int
4277 }
4278 { #1 }
4279 \@@_set_initial_coords:
4280 }
4281 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4282 {
4283 \pgfpointanchor
4284 {
4285 \@@_env:
4286 - \int_use:N \l_@@_final_i_int
4287 - \int_use:N \l_@@_final_j_int
4288 }
4289 { #1 }
4290 \@@_set_final_coords:
4291 }
4292 \cs_new_protected:Npn \@@_open_x_initial_dim:
4293 {
4294 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4295 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4296 {
4297 \cs_if_exist:cT
4298 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4299 {
4300 \pgfpointanchor
4301 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4302 { west }
4303 \dim_set:Nn \l_@@_x_initial_dim
4304 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4305 }
4306 }
4307 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4308 {
4309 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4310 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4311 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4312 }
4313 }
4314 \cs_new_protected:Npn \@@_open_x_final_dim:
4315 {
4316 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4317 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4318 {
4319 \cs_if_exist:cT
4320 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4321 {
4322 \pgfpointanchor
4323 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4324 { east }
4325 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4326 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4327 }

```

```
4328     }
```

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

```
4329     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4330     {
4331         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4332         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4333         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4334     }
4335 }
```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```
4336 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4337 {
4338     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4339     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4340     {
4341         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1
```

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

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

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

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

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`.

```
4356 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4357 {
4358     \bool_if:NTF \l_@@_initial_open_bool
4359     {
4360         \@@_open_x_initial_dim:
4361         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4362         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4363     }
4364     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4365 \bool_if:NTF \l_@@_final_open_bool
4366 {
4367   \@@_open_x_final_dim:
4368   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4369   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4370 }
4371 { \@@_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.

```

4372 \bool_lazy_all:nTF
4373 {
4374   \l_@@_initial_open_bool
4375   \l_@@_final_open_bool
4376   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4377 }
4378 {
4379   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4380   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4381 }

```

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 (?).

```

4382 {
4383   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4384   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4385 }
4386 \@@_draw_line:
4387 }

```

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.

```

4388 \cs_new_protected:Npn \@@_draw_Cdots:nmn #1 #2 #3
4389 {
4390   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4391   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4392   {
4393     \@@_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.

```

4394   \group_begin:
4395   \@@_open_shorten:
4396   \int_if_zero:nTF { #1 }
4397   { \color { nicematrix-first-row } }
4398   {

```

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.

```

4399     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4400     { \color { nicematrix-last-row } }
4401   }
4402   \keys_set:nn { nicematrix / xdots } { #3 }
4403   \@@_color:o \l_@@_xdots_color_tl
4404   \@@_actually_draw_Cdots:
4405 \group_end:
4406 }
4407 }

```

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.

```

4408 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4409 {
4410   \bool_if:NTF \l_@@_initial_open_bool
4411     { \@@_open_x_initial_dim: }
4412     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4413   \bool_if:NTF \l_@@_final_open_bool
4414     { \@@_open_x_final_dim: }
4415     { \@@_set_final_coords_from_anchor:n { mid-west } }
4416   \bool_lazy_and:mnTF
4417     \l_@@_initial_open_bool
4418     \l_@@_final_open_bool
4419     {
4420       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4421       \dim_set_eq:NN \l_tmpa_dim \pgf@y
4422       \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4423       \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4424       \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4425     }
4426   {
4427     \bool_if:NT \l_@@_initial_open_bool
4428       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4429     \bool_if:NT \l_@@_final_open_bool
4430       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4431   }
4432   \@@_draw_line:
4433 }
4434 \cs_new_protected:Npn \@@_open_y_initial_dim:
4435 {
4436   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4437   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4438     {
4439       \cs_if_exist:cT
4440         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4441         {
4442           \pgfpointanchor
4443             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4444             { north }
4445           \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4446             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4447         }
4448     }
4449   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4450     {
4451       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4452       \dim_set:Nn \l_@@_y_initial_dim
4453         {
4454           \fp_to_dim:n
4455             {
4456               \pgf@y
4457               + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4458             }
4459         }
4460     }
4461 }

```

```

4462 \cs_new_protected:Npn \l_@@_open_y_final_dim:
4463 {
4464   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4465   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4466   {
4467     \cs_if_exist:cT
4468     { \pgf @ sh @ ns @ \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4469     {
4470       \pgfpointanchor
4471       { \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4472       { south }
4473       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4474       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4475     }
4476   }
4477   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4478   {
4479     \l_@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4480     \dim_set:Nn \l_@@_y_final_dim
4481     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4482   }
4483 }

```

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.

```

4484 \cs_new_protected:Npn \l_@@_draw_Vdots:nnn #1 #2 #3
4485 {
4486   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4487   \cs_if_free:cT { \l_@@_dotted_#1 - #2 }
4488   {
4489     \l_@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 0

```

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

```

4490   \group_begin:
4491   \l_@@_open_shorten:
4492   \int_if_zero:nTF { #2 }
4493   { \color { nicematrix-first-col } }
4494   {
4495     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4496     { \color { nicematrix-last-col } }
4497   }
4498   \keys_set:nn { nicematrix / xdots } { #3 }
4499   \l_@@_color:o \l_@@_xdots_color_tl
4500   \l_@@_actually_draw_Vdots:
4501   \group_end:
4502 }
4503 }

```

The command `\l_@@_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`.

```

4504 \cs_new_protected:Npn \l_@@_actually_draw_Vdots:
4505 {

```

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

```
4506     \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.

```
4507     {
4508     \@@_open_y_initial_dim:
4509     \@@_open_y_final_dim:
4510     \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4511     {
4512     \@@_qpoint:n { col - 1 }
4513     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4514     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4515     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4516     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4517     }
4518     {
4519     \bool_lazy_and:nnTF
4520     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4521     { \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”.

```
4522     {
4523     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4524     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4525     \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4526     \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4527     \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4528     }
```

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

```
4529     {
4530     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4531     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4532     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4533     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4534     }
4535     }
4536 }
```

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

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```
4537     {
4538     \bool_set_false:N \l_tmpa_bool
4539     \bool_if:NF \l_@@_initial_open_bool
4540     {
4541     \bool_if:NF \l_@@_final_open_bool
4542     {
4543     \@@_set_initial_coords_from_anchor:n { south-west }
4544     \@@_set_final_coords_from_anchor:n { north-west }
4545     \bool_set:Nn \l_tmpa_bool
4546     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4547     }
4548     }
```

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

```
4549     \bool_if:NTF \l_@@_initial_open_bool
4550     {
4551     \@@_open_y_initial_dim:
4552     \@@_set_final_coords_from_anchor:n { north }
4553     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4554     }
4555     {
4556     \@@_set_initial_coords_from_anchor:n { south }
4557     \bool_if:NTF \l_@@_final_open_bool
```

```
4558         \l_@@_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.

```
4559         {
4560         \l_@@_set_final_coords_from_anchor:n { north }
4561         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4562         {
4563         \dim_set:Nn \l_@@_x_initial_dim
4564         {
4565         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4566         \l_@@_x_initial_dim \l_@@_x_final_dim
4567         }
4568         }
4569         }
4570     }
4571 }
4572 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4573 \l_@@_draw_line:
4574 }
```

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.

```
4575 \cs_new_protected:Npn \l_@@_draw_Ddots:nnn #1 #2 #3
4576 {
4577     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4578     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4579     {
4580         \l_@@_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.

```
4581     \group_begin:
4582     \l_@@_open_shorten:
4583     \keys_set:nn { nicematrix / xdots } { #3 }
4584     \l_@@_color:o \l_@@_xdots_color_tl
4585     \l_@@_actually_draw_Ddots:
4586     \group_end:
4587 }
4588 }
```

The command `\l_@@_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`.

```
4589 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4590 {
4591     \bool_if:NTF \l_@@_initial_open_bool
4592     {
4593         \l_@@_open_y_initial_dim:
4594         \l_@@_open_x_initial_dim:
```



```

4595     }
4596     { \@@_set_initial_coords_from_anchor:n { south-east } }
4597 \bool_if:NTF \l_@@_final_open_bool
4598     {
4599     \@@_open_x_final_dim:
4600     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4601     }
4602     { \@@_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.

```

4603     \bool_if:NT \l_@@_parallelize_diags_bool
4604     {
4605     \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).

```

4606     \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.

```

4607     {
4608     \dim_gset:Nn \g_@@_delta_x_one_dim
4609     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4610     \dim_gset:Nn \g_@@_delta_y_one_dim
4611     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4612     }

```

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`.

```

4613     {
4614     \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4615     {
4616     \dim_set:Nn \l_@@_y_final_dim
4617     {
4618     \l_@@_y_initial_dim +
4619     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4620     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4621     }
4622     }
4623     }
4624     }
4625 \@@_draw_line:
4626 }

```

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.

```

4627 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4628 {
4629 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4630 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4631 {
4632 \@@_find_extremities_of_line:nmmm { #1 } { #2 } 1 { -1 }

```

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

```

4633 \group_begin:
4634 \@@_open_shorten:
4635 \keys_set:nn { nicematrix / xdots } { #3 }
4636 \@@_color:o \l_@@_xdots_color_tl
4637 \@@_actually_draw_Iddots:
4638 \group_end:
4639 }
4640 }

```

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.`

```

4641 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4642   {
4643     \bool_if:NTF \l_@@_initial_open_bool
4644       {
4645         \@@_open_y_initial_dim:
4646         \@@_open_x_initial_dim:
4647       }
4648     { \@@_set_initial_coords_from_anchor:n { south-west } }
4649     \bool_if:NTF \l_@@_final_open_bool
4650       {
4651         \@@_open_y_final_dim:
4652         \@@_open_x_final_dim:
4653       }
4654     { \@@_set_final_coords_from_anchor:n { north-east } }
4655     \bool_if:NT \l_@@_parallelize_diags_bool
4656     {
4657       \int_gincr:N \g_@@_iddots_int
4658       \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4659         {
4660           \dim_gset:Nn \g_@@_delta_x_two_dim
4661             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4662           \dim_gset:Nn \g_@@_delta_y_two_dim
4663             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4664         }
4665         {
4666           \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4667             {
4668               \dim_set:Nn \l_@@_y_final_dim
4669                 {
4670                   \l_@@_y_initial_dim +
4671                   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4672                   \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4673                 }
4674             }
4675         }
4676     }
4677     \@@_draw_line:
4678   }

```

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`

```

4679 \cs_new_protected:Npn \@@_draw_line:
4680 {
4681   \pgfrememberpicturepositiononpagetrue
4682   \pgf@relevantforpicturesizefalse
4683   \bool_lazy_or:nnTF
4684     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4685     \l_@@_dotted_bool
4686     \@@_draw_standard_dotted_line:
4687     \@@_draw_unstandard_dotted_line:
4688 }

```

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.

```

4689 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4690 {
4691   \begin { scope }
4692     \@@_draw_unstandard_dotted_line:o
4693     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4694 }

```

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.

```

4695 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4696 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4697 {
4698   \@@_draw_unstandard_dotted_line:nooo
4699   { #1 }
4700   \l_@@_xdots_up_tl
4701   \l_@@_xdots_down_tl
4702   \l_@@_xdots_middle_tl
4703 }

```

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

```

4704 \hook_gput_code:nnn { begindocument } { . }
4705 {
4706   \IfPackageLoadedT { tikz }
4707   {
4708     \tikzset
4709     {
4710       @@_node_above / .style = { sloped , above } ,
4711       @@_node_below / .style = { sloped , below } ,
4712       @@_node_middle / .style =
4713       {
4714         sloped ,
4715         inner~sep = \c_@@_innersep_middle_dim
4716       }
4717     }
4718   }
4719 }

```

```

4720 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nmmn { n o o o }
4721 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nmmn #1 #2 #3 #4
4722 {

```

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.

```

4723 \dim_zero_new:N \l_@@_l_dim
4724 \dim_set:Nn \l_@@_l_dim
4725 {
4726 \fp_to_dim:n
4727 {
4728 sqrt
4729 (
4730 (\l_@@_x_final_dim - \l_@@_x_initial_dim) ^ 2
4731 +
4732 (\l_@@_y_final_dim - \l_@@_y_initial_dim) ^ 2
4733 )
4734 }
4735 }

```

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.

```

4736 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4737 {
4738 \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4739 \@@_draw_unstandard_dotted_line_i:
4740 }

```

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

```

4741 \bool_if:NT \l_@@_xdots_h_labels_bool
4742 {
4743 \tikzset
4744 {
4745 \@@_node_above / .style = { auto = left } ,
4746 \@@_node_below / .style = { auto = right } ,
4747 \@@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4748 }
4749 }
4750 \tl_if_empty:nF { #4 }
4751 { \tikzset { \@@_node_middle / .append-style = { fill = white } } }
4752 \draw
4753 [ #1 ]
4754 ( \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).

```

4755 -- node [ \@@_node_middle ] { $ \scriptstyle #4 $ }
4756 node [ \@@_node_below ] { $ \scriptstyle #3 $ }
4757 node [ \@@_node_above ] { $ \scriptstyle #2 $ }
4758 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4759 \end { scope }
4760 }
4761 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4762 {
4763 \dim_set:Nn \l_tmpa_dim
4764 {
4765 \l_@@_x_initial_dim
4766 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4767     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4768   }
4769   \dim_set:Nn \l_tmpb_dim
4770   {
4771     \l_@@_y_initial_dim
4772     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4773     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4774   }
4775   \dim_set:Nn \l_@@_tmpc_dim
4776   {
4777     \l_@@_x_final_dim
4778     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4779     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4780   }
4781   \dim_set:Nn \l_@@_tmpd_dim
4782   {
4783     \l_@@_y_final_dim
4784     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4785     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4786   }
4787   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4788   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4789   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4790   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4791 }

```

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

```

4792 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4793 {
4794   \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.

```

4795   \dim_zero_new:N \l_@@_l_dim
4796   \dim_set:Nn \l_@@_l_dim
4797   {
4798     \fp_to_dim:n
4799     {
4800       sqrt
4801       (
4802         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4803         +
4804         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4805       )
4806     }
4807   }

```

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.

```

4808   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4809   {
4810     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4811     \@@_draw_standard_dotted_line_i:
4812   }
4813   \group_end:
4814   \bool_lazy_all:nF
4815   {
4816     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4817     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4818     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4819     }
4820     \l_@@_labels_standard_dotted_line:
4821 }
4822 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4823 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4824 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

4825     \int_set:Nn \l_1tmpa_int
4826     {
4827         \dim_ratio:nn
4828         {
4829             \l_@@_l_dim
4830             - \l_@@_xdots_shorten_start_dim
4831             - \l_@@_xdots_shorten_end_dim
4832         }
4833         \l_@@_xdots_inter_dim
4834     }

```

The dimensions $\l_1\text{tmpa_dim}$ and $\l_1\text{tmpb_dim}$ are the coordinates of the vector between two dots in the dotted line.

```

4835     \dim_set:Nn \l_1tmpa_dim
4836     {
4837         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4838         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4839     }
4840     \dim_set:Nn \l_1tmpb_dim
4841     {
4842         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4843         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4844     }

```

In the loop over the dots, the dimensions $\l_1\text{@@_x_initial_dim}$ and $\l_1\text{@@_y_initial_dim}$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4845     \dim_gadd:Nn \l_@@_x_initial_dim
4846     {
4847         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4848         \dim_ratio:nn
4849         {
4850             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4851             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4852         }
4853         { 2 \l_@@_l_dim }
4854     }
4855     \dim_gadd:Nn \l_@@_y_initial_dim
4856     {
4857         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4858         \dim_ratio:nn
4859         {
4860             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4861             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4862         }
4863         { 2 \l_@@_l_dim }
4864     }
4865     \pgf@relevantforpicturesizefalse
4866     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4867     {
4868         \pgfpathcircle
4869         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4870         { \l_@@_xdots_radius_dim }
4871         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4872         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4873     }

```

```

4874 \pgfusepathqfill
4875 }

4876 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4877 {
4878 \pgfscope
4879 \pgftransformshift
4880 {
4881 \pgfpointlineattime { 0.5 }
4882 { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4883 { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4884 }
4885 \fp_set:Nn \l_tmpa_fp
4886 {
4887 atand
4888 (
4889 \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4890 \l_@@_x_final_dim - \l_@@_x_initial_dim
4891 )
4892 }
4893 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4894 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4895 \tl_if_empty:NF \l_@@_xdots_middle_tl
4896 {
4897 \begin { pgfscope }
4898 \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4899 \pgfnode
4900 { rectangle }
4901 { center }
4902 {
4903 \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4904 {
4905 \c_math_toggle_token
4906 \scriptstyle \l_@@_xdots_middle_tl
4907 \c_math_toggle_token
4908 }
4909 }
4910 { }
4911 {
4912 \pgfsetfillcolor { white }
4913 \pgfusepath { fill }
4914 }
4915 \end { pgfscope }
4916 }
4917 \tl_if_empty:NF \l_@@_xdots_up_tl
4918 {
4919 \pgfnode
4920 { rectangle }
4921 { south }
4922 {
4923 \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4924 {
4925 \c_math_toggle_token
4926 \scriptstyle \l_@@_xdots_up_tl
4927 \c_math_toggle_token
4928 }
4929 }
4930 { }
4931 { \pgfusepath { } }
4932 }
4933 \tl_if_empty:NF \l_@@_xdots_down_tl
4934 {
4935 \pgfnode

```

```

4936     { rectangle }
4937     { north }
4938     {
4939       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4940       {
4941         \c_math_toggle_token
4942         \scriptstyle \l_@@_xdots_down_tl
4943         \c_math_toggle_token
4944       }
4945     }
4946     { }
4947     { \pgfusepath { } }
4948   }
4949 \endpgfscope
4950 }

```

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.

```

4951 \hook_gput_code:nnn { begindocument } { . }
4952 {
4953   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4954   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4955   \cs_new_protected:Npn \@@_Ldots
4956     { \@@_collect_options:n { \@@_Ldots_i } }
4957   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4958     {
4959     \int_if_zero:nTF \c@jCol
4960       { \@@_error:nn { in~first~col } \Ldots }
4961       {
4962         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4963           { \@@_error:nn { in~last~col } \Ldots }
4964           {
4965             \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4966             { #1 , down = #2 , up = #3 , middle = #4 }
4967           }
4968         }
4969     \bool_if:NF \l_@@_nullify_dots_bool
4970       { \phantom { \ensuremath { \@@_old_ldots } } }
4971     \bool_gset_true:N \g_@@_empty_cell_bool
4972   }

4973 \cs_new_protected:Npn \@@_Cdots
4974   { \@@_collect_options:n { \@@_Cdots_i } }
4975 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4976   {
4977   \int_if_zero:nTF \c@jCol
4978     { \@@_error:nn { in~first~col } \Cdots }
4979     {
4980       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```



```

4981         { \@@_error:nn { in~last~col } \Cdots }
4982         {
4983             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4984             { #1 , down = #2 , up = #3 , middle = #4 }
4985         }
4986     }
4987     \bool_if:NF \l_@@_nullify_dots_bool
4988     { \phantom { \ensuremath { \@@_old_cdots } } }
4989     \bool_gset_true:N \g_@@_empty_cell_bool
4990 }

4991 \cs_new_protected:Npn \@@_Vdots
4992 { \@@_collect_options:n { \@@_Vdots_i } }
4993 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4994 {
4995     \int_if_zero:nTF \c@iRow
4996     { \@@_error:nn { in~first~row } \Vdots }
4997     {
4998         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4999         { \@@_error:nn { in~last~row } \Vdots }
5000         {
5001             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5002             { #1 , down = #2 , up = #3 , middle = #4 }
5003         }
5004     }
5005     \bool_if:NF \l_@@_nullify_dots_bool
5006     { \phantom { \ensuremath { \@@_old_vdots } } }
5007     \bool_gset_true:N \g_@@_empty_cell_bool
5008 }

5009 \cs_new_protected:Npn \@@_Ddots
5010 { \@@_collect_options:n { \@@_Ddots_i } }
5011 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5012 {
5013     \int_case:nnF \c@iRow
5014     {
5015         0 { \@@_error:nn { in~first~row } \Ddots }
5016         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5017     }
5018     {
5019         \int_case:nnF \c@jCol
5020         {
5021             0 { \@@_error:nn { in~first~col } \Ddots }
5022             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5023         }
5024         {
5025             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5026             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5027             { #1 , down = #2 , up = #3 , middle = #4 }
5028         }
5029     }
5030 }
5031 \bool_if:NF \l_@@_nullify_dots_bool
5032 { \phantom { \ensuremath { \@@_old_ddots } } }
5033 \bool_gset_true:N \g_@@_empty_cell_bool
5034 }

5035 \cs_new_protected:Npn \@@_Iddots
5036 { \@@_collect_options:n { \@@_Iddots_i } }
5037 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5038 {

```

```

5039     \int_case:nnF \c@iRow
5040     {
5041         0           { \@@_error:nn { in~first~row } \Iddots }
5042         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5043     }
5044     {
5045         \int_case:nnF \c@jCol
5046         {
5047             0           { \@@_error:nn { in~first~col } \Iddots }
5048             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5049         }
5050         {
5051             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5052             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5053             { #1 , down = #2 , up = #3 , middle = #4 }
5054         }
5055     }
5056     \bool_if:NF \l_@@_nullify_dots_bool
5057     { \phantom { \ensuremath { \@@_old_iddots } } }
5058     \bool_gset_true:N \g_@@_empty_cell_bool
5059 }
5060 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5061 \keys_define:nn { nicematrix / Ddots }
5062 {
5063     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5064     draw-first .default:n = true ,
5065     draw-first .value_forbidden:n = true
5066 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5067 \cs_new_protected:Npn \@@_Hspace:
5068 {
5069     \bool_gset_true:N \g_@@_empty_cell_bool
5070     \hspace
5071 }

```

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`.

```

5072 \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`.

```

5073 \cs_new:Npn \@@_Hdotsfor:
5074 {
5075     \bool_lazy_and:nnTF
5076     { \int_if_zero_p:n \c@jCol }
5077     { \int_if_zero_p:n \l_@@_first_col_int }
5078     {
5079         \bool_if:NTF \g_@@_after_col_zero_bool
5080         {
5081             \multicolumn { 1 } { c } { }
5082             \@@_Hdotsfor_i
5083         }
5084         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5085     }
5086 }

```

```

5087     \multicolumn { 1 } { c } { }
5088     \@@_Hdotsfor_i
5089   }
5090 }

```

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:`).

```

5091 \hook_gput_code:nnn { begindocument } { . }
5092 {
5093   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5094   \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.

```

5095   \cs_new_protected:Npn \@@_Hdotsfor_i
5096     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5097   \exp_args:NNno \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5098     {
5099     \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5100       {
5101         \@@_Hdotsfor:nnnn
5102         { \int_use:N \c@iRow }
5103         { \int_use:N \c@jCol }
5104         { #2 }
5105         {
5106           #1 , #3 ,
5107           down = \exp_not:n { #4 } ,
5108           up = \exp_not:n { #5 } ,
5109           middle = \exp_not:n { #6 }
5110         }
5111       }
5112     \prg_replicate:nn { #2 - 1 }
5113     {
5114       &
5115       \multicolumn { 1 } { c } { }
5116       \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5117     }
5118   }
5119 }

```

```

5120 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5121 {
5122   \bool_set_false:N \l_@@_initial_open_bool
5123   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5124   \int_set:Nn \l_@@_initial_i_int { #1 }
5125   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5126   \int_compare:nNnTF { #2 } = \c_one_int
5127     {
5128     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5129     \bool_set_true:N \l_@@_initial_open_bool
5130   }
5131   {
5132     \cs_if_exist:cTF
5133       {
5134         pgf @ sh @ ns @ \@@_env:
5135         - \int_use:N \l_@@_initial_i_int
5136         - \int_eval:n { #2 - 1 }
5137       }
5138     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5139     {

```

```

5140         \int_set:Nn \l_@@_initial_j_int { #2 }
5141         \bool_set_true:N \l_@@_initial_open_bool
5142     }
5143 }
5144 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5145 {
5146     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5147     \bool_set_true:N \l_@@_final_open_bool
5148 }
5149 {
5150     \cs_if_exist:cTF
5151     {
5152         pgf @ sh @ ns @ \@@_env:
5153         - \int_use:N \l_@@_final_i_int
5154         - \int_eval:n { #2 + #3 }
5155     }
5156     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5157     {
5158         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5159         \bool_set_true:N \l_@@_final_open_bool
5160     }
5161 }
5162 \group_begin:
5163 \@@_open_shorten:
5164 \int_if_zero:nTF { #1 }
5165 { \color { nicematrix-first-row } }
5166 {
5167     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5168     { \color { nicematrix-last-row } }
5169 }
5170
5171 \keys_set:nn { nicematrix / xdots } { #4 }
5172 \@@_color:o \l_@@_xdots_color_tl
5173 \@@_actually_draw_Ldots:
5174 \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).

```

5175     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5176     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5177 }

5178 \hook_gput_code:nnn { begindocument } { . }
5179 {
5180     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5181     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5182     \cs_new_protected:Npn \@@_Vdotsfor:
5183     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5184     \exp_args:NNNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5185     {
5186         \bool_gset_true:N \g_@@_empty_cell_bool
5187         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5188         {
5189             \@@_Vdotsfor:nnnn
5190             { \int_use:N \c@iRow }
5191             { \int_use:N \c@jCol }
5192             { #2 }
5193             {
5194                 #1 , #3 ,
5195                 down = \exp_not:n { #4 } ,
5196                 up = \exp_not:n { #5 } ,

```

```

5197         middle = \exp_not:n { #6 }
5198     }
5199 }
5200 }
5201 }

```

```

5202 \cs_new_protected:Npn \l_@@_Vdotsfor:nmmn #1 #2 #3 #4
5203 {
5204     \bool_set_false:N \l_@@_initial_open_bool
5205     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5206     \int_set:Nn \l_@@_initial_j_int { #2 }
5207     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5208     \int_compare:nNnTF { #1 } = \c_one_int
5209     {
5210         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5211         \bool_set_true:N \l_@@_initial_open_bool
5212     }
5213     {
5214         \cs_if_exist:cTF
5215         {
5216             pgf @ sh @ ns @ \l_@@_env:
5217             - \int_eval:n { #1 - 1 }
5218             - \int_use:N \l_@@_initial_j_int
5219         }
5220         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5221         {
5222             \int_set:Nn \l_@@_initial_i_int { #1 }
5223             \bool_set_true:N \l_@@_initial_open_bool
5224         }
5225     }
5226     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5227     {
5228         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5229         \bool_set_true:N \l_@@_final_open_bool
5230     }
5231     {
5232         \cs_if_exist:cTF
5233         {
5234             pgf @ sh @ ns @ \l_@@_env:
5235             - \int_eval:n { #1 + #3 }
5236             - \int_use:N \l_@@_final_j_int
5237         }
5238         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5239         {
5240             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5241             \bool_set_true:N \l_@@_final_open_bool
5242         }
5243     }
5244     \group_begin:
5245     \l_@@_open_shorten:
5246     \int_if_zero:nTF { #2 }
5247     { \color { nicematrix-first-col } }
5248     {
5249         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5250         { \color { nicematrix-last-col } }
5251     }
5252     \keys_set:nn { nicematrix / xdots } { #4 }
5253     \l_@@_color:o \l_@@_xdots_color_tl
5254     \l_@@_actually_draw_Vdots:
5255     \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:nnn`). This declaration is done by defining a special control sequence (to nil).

```

5256 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5257   { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5258 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5259 \NewDocumentCommand \@@_rotate: { 0 { } }
5260 {
5261   \peek_remove_spaces:n
5262   {
5263     \bool_gset_true:N \g_@@_rotate_bool
5264     \keys_set:nn { nicematrix / rotate } { #1 }
5265   }
5266 }

5267 \keys_define:nn { nicematrix / rotate }
5268 {
5269   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5270   c .value_forbidden:n = true ,
5271   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5272 }

```

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

```

5273 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5274 {
5275   \tl_if_empty:nTF { #2 }
5276     { #1 }
5277     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5278 }
5279 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5280 { \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).

```

5281 \hook_gput_code:nnn { begindocument } { . }
5282 {

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5283 \cs_set_nopar:Npn \l_@@_argspec_tl
5284   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5285 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5286 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5287   {
5288     \group_begin:
5289     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5290     \@@_color:o \l_@@_xdots_color_tl
5291     \use:e
5292     {
5293       \@@_line_i:nn
5294       { \@@_double_int_eval:n #2 - \q_stop }
5295       { \@@_double_int_eval:n #3 - \q_stop }
5296     }
5297     \group_end:
5298   }
5299 }

5300 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5301   {
5302     \bool_set_false:N \l_@@_initial_open_bool
5303     \bool_set_false:N \l_@@_final_open_bool
5304     \bool_lazy_or:nnTF
5305       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5306       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5307     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5308   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5309 }

5310 \hook_gput_code:nnn { begindocument } { . }
5311 {
5312   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5313   {

```

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:.`

```

5314     \c_@@_pgfortikzpicture_tl
5315     \@@_draw_line_iii:nn { #1 } { #2 }
5316     \c_@@_endpgfortikzpicture_tl
5317   }
5318 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5319 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5320   {
5321     \pgfrememberpicturepositiononpagetrue
5322     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5323     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5324     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5325     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5326     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5327     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5328     \@@_draw_line:
5329   }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

```
5330 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5331   { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5332 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5333 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5334   {
5335     \tl_gput_right:Ne \g_@@_row_style_tl
5336     {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5337     \exp_not:N
5338     \@@_if_row_less_than:nn
5339     { \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`).

```
5340     { \exp_not:n { #1 } \scan_stop: }
5341   }
5342 }
```

```
5343 \keys_define:nn { nicematrix / RowStyle }
5344   {
5345     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5346     cell-space-top-limit .value_required:n = true ,
5347     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5348     cell-space-bottom-limit .value_required:n = true ,
5349     cell-space-limits .meta:n =
5350     {
5351       cell-space-top-limit = #1 ,
5352       cell-space-bottom-limit = #1 ,
5353     } ,
5354     color .tl_set:N = \l_@@_color_tl ,
5355     color .value_required:n = true ,
5356     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5357     bold .default:n = true ,
5358     nb-rows .code:n =
5359     \str_if_eq:eeTF { #1 } { * }
5360     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5361     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5362     nb-rows .value_required:n = true ,
5363     fill .tl_set:N = \l_@@_fill_tl ,
5364     fill .value_required:n = true ,
5365     opacity .tl_set:N = \l_@@_opacity_tl ,
5366     opacity .value_required:n = true ,
5367     rowcolor .tl_set:N = \l_@@_fill_tl ,
5368     rowcolor .value_required:n = true ,
5369     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
```



```

5370     rounded-corners .default:n = 4 pt ,
5371     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5372 }

```

```

5373 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5374 {
5375   \group_begin:
5376   \tl_clear:N \l_@@_fill_tl
5377   \tl_clear:N \l_@@_opacity_tl
5378   \tl_clear:N \l_@@_color_tl
5379   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5380   \dim_zero:N \l_@@_rounded_corners_dim
5381   \dim_zero:N \l_tmpa_dim
5382   \dim_zero:N \l_tmpb_dim
5383   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key rowcolor (of its alias fill) has been used.

```

5384   \tl_if_empty:NF \l_@@_fill_tl
5385   {
5386     \@@_add_opacity_to_fill:
5387     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5388     {

```

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.

```

5389       \int_compare:nNnTF \c@jCol > \c_one_int
5390       {

```

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*.

```

5391         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5392         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5393         { \int_use:N \c@iRow - * }
5394         { \dim_use:N \l_@@_rounded_corners_dim }

```

Then, the other rows (if there are several rows).

```

5395         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5396         { \@@_rounded_from_row:n { \c@iRow + 1 } }
5397     }

```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```

5398     { \@@_rounded_from_row:n { \c@iRow } }
5399   }
5400 }
5401 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5402   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5403   {
5404     \@@_put_in_row_style:e
5405     {
5406       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5407       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5408         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5409         { \dim_use:N \l_tmpa_dim }
5410     }
5411 }
5412 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5413   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5414   {
5415     \@@_put_in_row_style:e
5416     {
5417       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5418       {
5419         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5420         { \dim_use:N \l_tmpb_dim }
5421       }
5422     }
5423   }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5424   \tl_if_empty:NF \l_@@_color_tl
5425   {
5426     \@@_put_in_row_style:e
5427     {
5428       \mode_leave_vertical:
5429       \@@_color:n { \l_@@_color_tl }
5430     }
5431   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5432   \bool_if:NT \l_@@_bold_row_style_bool
5433   {
5434     \@@_put_in_row_style:n
5435     {
5436       \exp_not:n
5437       {
5438         \if_mode_math:
5439         \c_math_toggle_token
5440         \bfseries \boldmath
5441         \c_math_toggle_token
5442         \else:
5443         \bfseries \boldmath
5444         \fi:
5445       }
5446     }
5447   }
5448   \group_end:
5449   \g_@@_row_style_tl
5450   \ignorespaces
5451 }

```

The following commande must *not* be protected.

```

5452 \cs_new:Npn \@@_rounded_from_row:n #1
5453 {
5454   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5455   { \int_eval:n { #1 } - 1 }
5456   {
5457     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5458     - \exp_not:n { \int_use:N \c@jCol }
5459   }
5460   { \dim_use:N \l_@@_rounded_corners_dim }
5461 }

```

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).

```
5462 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5463 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5464 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5465 {
```

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.

```
5466 \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`.

```
5467 \str_if_in:nnF { #1 } { !! }
5468 {
5469 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5470 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5471 }
5472 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5473 {
5474 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5475 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5476 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5477 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5478 }
```

The following command must be used within a `\pgfpicture`.

```
5479 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5480 {
5481 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5482 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5483     \group_begin:
5484     \pgfsetcornersarced
5485     {
5486         \pgfpoint
5487         { \l_@@_tab_rounded_corners_dim }
5488         { \l_@@_tab_rounded_corners_dim }
5489     }

```

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.

```

5490     \bool_if:NTF \l_@@_hvlines_bool
5491     {
5492         \pgfpathrectanglecorners
5493         {
5494             \pgfpointadd
5495             { \@@_qpoint:n { row-1 } }
5496             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5497         }
5498         {
5499             \pgfpointadd
5500             {
5501                 \@@_qpoint:n
5502                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5503             }
5504             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5505         }
5506     }
5507     {
5508         \pgfpathrectanglecorners
5509         { \@@_qpoint:n { row-1 } }
5510         {
5511             \pgfpointadd
5512             {
5513                 \@@_qpoint:n
5514                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5515             }
5516             { \pgfpoint \c_zero_dim \arrayrulewidth }
5517         }
5518     }
5519     \pgfusepath { clip }
5520     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5521     }
5522 }

```

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`).

```

5523 \cs_new_protected:Npn \@@_actually_color:
5524 {
5525     \pgfpicture
5526     \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.

```

5527     \@@_clip_with_rounded_corners:
5528     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5529     {
5530         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5531     {
5532     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5533     \use:c { g_@@_color _ 1 _tl }
5534     \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5535     }
5536     {
5537     \begin { pgfscope }
5538     \@@_color_opacity ##2
5539     \use:c { g_@@_color _ ##1 _tl }
5540     \tl_gclear:c { g_@@_color _ ##1 _tl }
5541     \pgfusepath { fill }
5542     \end { pgfscope }
5543     }
5544     }
5545 \endpgfpicture
5546 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5547 \cs_new_protected:Npn \@@_color_opacity
5548 {
5549   \peek_meaning:NTF [
5550     { \@@_color_opacity:w }
5551     { \@@_color_opacity:w [ ] }
5552   }

```

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.

```

5553 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5554 {
5555   \tl_clear:N \l_tmpa_tl
5556   \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.

```

5557   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5558   \tl_if_empty:NTF \l_tmpb_tl
5559     { \@declaredcolor }
5560     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5561 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5562 \keys_define:nn { nicematrix / color-opacity }
5563 {
5564   opacity .tl_set:N          = \l_tmpa_tl ,
5565   opacity .value_required:n = true
5566 }

5567 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5568 {
5569   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5570   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5571   \@@_cartesian_path:
5572 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5573 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5574 {
5575   \tl_if_blank:nF { #2 }
5576   {

```

```

5577     \@@_add_to_colors_seq:en
5578     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5579     { \@@_cartesian_color:nn { #3 } { - } }
5580   }
5581 }

```

Here an example : `\@@_columncolor{red!15}{1,3,5-7,10-}`

```

5582 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5583 {
5584   \tl_if_blank:nF { #2 }
5585   {
5586     \@@_add_to_colors_seq:en
5587     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5588     { \@@_cartesian_color:nn { - } { #3 } }
5589   }
5590 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5591 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5592 {
5593   \tl_if_blank:nF { #2 }
5594   {
5595     \@@_add_to_colors_seq:en
5596     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5597     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5598   }
5599 }

```

The last argument is the radius of the corners of the rectangle.

```

5600 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5601 {
5602   \tl_if_blank:nF { #2 }
5603   {
5604     \@@_add_to_colors_seq:en
5605     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5606     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5607   }
5608 }

```

The last argument is the radius of the corners of the rectangle.

```

5609 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5610 {
5611   \@@_cut_on_hyphen:w #1 \q_stop
5612   \tl_clear_new:N \l_@@_tmpc_tl
5613   \tl_clear_new:N \l_@@_tmpd_tl
5614   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5615   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5616   \@@_cut_on_hyphen:w #2 \q_stop
5617   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5618   \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`.

```

5619   \@@_cartesian_path:n { #3 }
5620 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5621 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5622 {
5623   \clist_map_inline:nn { #3 }
5624   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5625 }

```

```

5626 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5627 {
5628   \int_step_inline:nn \c@iRow
5629   {
5630     \int_step_inline:nn \c@jCol
5631     {
5632       \int_if_even:nTF { ####1 + #1 }
5633       { \@@_cellcolor [ #1 ] { #2 } }
5634       { \@@_cellcolor [ #1 ] { #3 } }
5635       { #1 - ####1 }
5636     }
5637   }
5638 }

```

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”.

```

5639 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5640 {
5641   \@@_rectanglecolor [ #1 ] { #2 }
5642   { 1 - 1 }
5643   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5644 }

```

```

5645 \keys_define:nn { nicematrix / rowcolors }
5646 {
5647   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5648   respect-blocks .default:n = true ,
5649   cols .tl_set:N = \l_@@_cols_tl ,
5650   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5651   restart .default:n = true ,
5652   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5653 }

```

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*.

```

5654 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5655 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5656   \group_begin:
5657   \seq_clear_new:N \l_@@_colors_seq
5658   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5659   \tl_clear_new:N \l_@@_cols_tl
5660   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5661   \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).

```

5662   \int_zero_new:N \l_@@_color_int
5663   \int_set_eq:NN \l_@@_color_int \c_one_int
5664   \bool_if:NT \l_@@_respect_blocks_bool
5665   {

```

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`).

```

5666     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5667     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5668     { \@@_not_in_exterior_p:nnnnn ##1 }
5669   }
5670   \pgfpicture
5671   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5672   \clist_map_inline:nn { #2 }
5673   {
5674     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5675     \tl_if_in:NnTF \l_tmpa_tl { - }
5676     { \@@_cut_on_hyphen:w ##1 \q_stop }
5677     { \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.

```

5678     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5679     \int_set:Nn \l_@@_color_int
5680     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5681     \int_zero_new:N \l_@@_tmpc_int
5682     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5683     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5684     {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5685     \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).

```

5686     \bool_if:NT \l_@@_respect_blocks_bool
5687     {
5688       \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5689       { \@@_intersect_our_row_p:nnnnn #####1 }
5690       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5691     }
5692     \tl_set:No \l_@@_rows_tl
5693     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5694     \tl_clear_new:N \l_@@_color_tl
5695     \tl_set:Ne \l_@@_color_tl
5696     {
5697       \@@_color_index:n
5698       {
5699         \int_mod:nn
5700         { \l_@@_color_int - 1 }
5701         { \seq_count:N \l_@@_colors_seq }
5702         + 1
5703       }
5704     }
5705     \tl_if_empty:NF \l_@@_color_tl
5706     {
5707       \@@_add_to_colors_seq:ee
5708       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5709       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5710     }
5711     \int_incr:N \l_@@_color_int
5712     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5713   }
5714 }
5715 \endpgfpicture

```



```

5716   \group_end:
5717 }

```

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.

```

5718 \cs_new:Npn \@@_color_index:n #1
5719 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5720   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5721   { \@@_color_index:n { #1 - 1 } }
5722   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5723 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5724 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5725 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5726 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5727 {
5728   \int_compare:nNnT { #3 } > \l_tmpb_int
5729   { \int_set:Nn \l_tmpb_int { #3 } }
5730 }

```

```

5731 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5732 {
5733   \int_if_zero:nTF { #4 }
5734   \prg_return_false:
5735   {
5736     \int_compare:nNnTF { #2 } > \c@jCol
5737     \prg_return_false:
5738     \prg_return_true:
5739   }
5740 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5741 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5742 {
5743   \int_compare:nNnTF { #1 } > \l_tmpa_int
5744   \prg_return_false:
5745   {
5746     \int_compare:nNnTF \l_tmpa_int > { #3 }
5747     \prg_return_false:
5748     \prg_return_true:
5749   }
5750 }

```

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`).

```

5751 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5752 {
5753   \dim_compare:nNnTF { #1 } = \c_zero_dim

```

```

5754 {
5755   \bool_if:NTF
5756     \l_@@_nocolor_used_bool
5757     \@@_cartesian_path_normal_ii:
5758     {
5759       \clist_if_empty:NTF \l_@@_corners_cells_clist
5760         { \@@_cartesian_path_normal_i:n { #1 } }
5761         \@@_cartesian_path_normal_ii:
5762     }
5763   }
5764   { \@@_cartesian_path_normal_i:n { #1 } }
5765 }

```

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.

```

5766 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5767 {
5768   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5769   \clist_map_inline:Nn \l_@@_cols_tl
5770   {
5771     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5772     \tl_if_in:NnTF \l_tmpa_tl { - }
5773       { \@@_cut_on_hyphen:w ##1 \q_stop }
5774       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5775     \tl_if_empty:NTF \l_tmpa_tl
5776       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5777     {
5778       \str_if_eq:eeT \l_tmpa_tl { * }
5779       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5780     }
5781     \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5782       { \@@_error:n { Invalid-col-number } }
5783     \tl_if_empty:NTF \l_tmpb_tl
5784       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5785     {
5786       \str_if_eq:eeT \l_tmpb_tl { * }
5787       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5788     }
5789     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5790     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5791   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5792   \@@_qpoint:n { col - \l_tmpa_tl }
5793   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5794     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5795     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5796   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5797   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5798   \clist_map_inline:Nn \l_@@_rows_tl
5799   {
5800     \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5801     \tl_if_in:NnTF \l_tmpa_tl { - }
5802       { \@@_cut_on_hyphen:w ####1 \q_stop }
5803       { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5804     \tl_if_empty:NTF \l_tmpa_tl
5805       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5806     {
5807       \str_if_eq:eeT \l_tmpa_tl { * }
5808       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }

```

```

5809     }
5810     \tl_if_empty:NTF \l_tmpb_tl
5811     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5812     {
5813         \str_if_eq:eeT \l_tmpb_tl { * }
5814         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5815     }
5816     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5817     { \@@_error:n { Invalid-row-number } }
5818     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5819     { \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`.

```

5820     \cs_if_exist:cF
5821     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5822     {
5823         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5824         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5825         \@@_qpoint:n { row - \l_tmpa_tl }
5826         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5827         \pgfpathrectanglecorners
5828         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5829         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5830     }
5831 }
5832 }
5833 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5834 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5835 {
5836     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5837     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5838     \clist_map_inline:Nn \l_@@_cols_tl
5839     {
5840         \@@_qpoint:n { col - ##1 }
5841         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5842         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5843         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5844         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5845         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5846     \clist_map_inline:Nn \l_@@_rows_tl
5847     {
5848         \@@_if_in_corner:nF { #####1 - ##1 }
5849         {
5850             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5851             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5852             \@@_qpoint:n { row - #####1 }
5853             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5854             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5855             {
5856                 \pgfpathrectanglecorners
5857                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5858                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5859             }
5860         }
5861     }
5862 }
5863 }

```

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`).

```
5864 \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

```
5865 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5866 {
5867   \bool_set_true:N \l_@@_nocolor_used_bool
5868   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5869   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5870   \clist_map_inline:Nn \l_@@_rows_tl
5871     {
5872       \clist_map_inline:Nn \l_@@_cols_tl
5873         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5874     }
5875 }
```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to `2,4-6,8-*` and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by `2,4,5,6,8,9,10`.

```
5876 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5877 {
5878   \clist_set_eq:NN \l_tmpa_clist #1
5879   \clist_clear:N #1
5880   \clist_map_inline:Nn \l_tmpa_clist
5881     {
5882       \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5883       \tl_if_in:NnTF \l_tmpa_tl { - }
5884         { \@@_cut_on_hyphen:w ##1 \q_stop }
5885         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5886       \bool_lazy_or:nnT
5887         { \str_if_eq_p:ee \l_tmpa_tl { * } }
5888         { \tl_if_blank_p:o \l_tmpa_tl }
5889         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5890       \bool_lazy_or:nnT
5891         { \str_if_eq_p:ee \l_tmpb_tl { * } }
5892         { \tl_if_blank_p:o \l_tmpb_tl }
5893         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5894       \int_compare:nNnT \l_tmpb_tl > #2
5895         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5896       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5897         { \clist_put_right:Nn #1 { ####1 } }
5898     }
5899 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5900 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5901 {
5902   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5903   {
```

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`).

```
5904     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5905     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5906   }
5907   \ignorespaces
5908 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

5909 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5910 {
5911   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5912   {
5913     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5914     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5915     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5916   }
5917   \ignorespaces
5918 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5919 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5920 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5921 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5922 {
5923   \peek_remove_spaces:n
5924   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5925 }

```

```

5926 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5927 {

```

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`.

```

5928   \seq_gclear:N \g_tmpa_seq
5929   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5930   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5931   \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`.

```

5932   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5933   {
5934     { \int_use:N \c@iRow }
5935     { \exp_not:n { #1 } }
5936     { \exp_not:n { #2 } }
5937     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5938   }
5939 }

```

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`).

```

5940 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5941 {
5942   \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).

```

5943     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5944     {
5945     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5946     {
5947     \@@_rowlistcolors
5948     [ \exp_not:n { #2 } ]
5949     { #1 - \int_eval:n { \c@iRow - 1 } }
5950     { \exp_not:n { #3 } }
5951     [ \exp_not:n { #4 } ]
5952     }
5953     }
5954 }

```

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.

```

5955 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5956 {
5957   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5958   { \@@_rowlistcolors_tabular_ii:nmmm ##1 }
5959   \seq_gclear:N \g_@@_rowlistcolors_seq
5960 }

5961 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nmmm #1 #2 #3 #4
5962 {
5963   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5964   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5965 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5966 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5967 {

```

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).

```

5968   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5969   {

```

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).

```

5970     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5971     {
5972     \exp_not:N \columncolor [ #1 ]
5973     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5974     }
5975     }
5976 }

```

```

5977 \hook_gput_code:nnn { begindocument } { . }
5978 {
5979   \IfPackageLoadedTF { colortbl }
5980   {
5981     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5982     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5983     \cs_new_protected:Npn \@@_revert_colortbl:

```

```

5984     {
5985         \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5986         {
5987             \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5988             \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5989         }
5990     }
5991 }
5992 { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5993 }

5994 \cs_new_protected:Npn \@@_EmptyColumn:n #1
5995 {
5996     \clist_map_inline:nn { #1 }
5997     {
5998         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5999         { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
6000         \columncolor { nocolor } { ##1 }
6001     }
6002 }

6003 \cs_new_protected:Npn \@@_EmptyRow:n #1
6004 {
6005     \clist_map_inline:nn { #1 }
6006     {
6007         \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
6008         { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
6009         \rowcolor { nocolor } { ##1 }
6010     }
6011 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumn` type 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`.

```
6012 \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`.

```

6013 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6014 {
6015     \int_if_zero:nTF \l_@@_first_col_int
6016     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6017     {
6018         \int_if_zero:nTF \c@jCol
6019         {
6020             \int_compare:nNf \c@iRow = { -1 }
6021             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6022         }
6023         { \@@_OnlyMainNiceMatrix_i:n { #1 } }

```

```

6024     }
6025 }

```

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.

```

6026 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6027 {
6028   \int_if_zero:nF \c@iRow
6029   {
6030     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6031     {
6032       \int_compare:nNnT \c@jCol > \c_zero_int
6033       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6034     }
6035   }
6036 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6037 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
6038 {
6039   \IfPackageLoadedTF { tikz }
6040   {
6041     \IfPackageLoadedTF { booktabs }
6042     { #2 }
6043     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6044   }
6045   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6046 }
6047 \NewExpandableDocumentCommand { \@@_TopRule } { }
6048 { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }
6049 \cs_new:Npn \@@_TopRule_i:
6050 {
6051   \noalign \bgroup
6052   \peek_meaning:NTF [
6053   { \@@_TopRule_ii: }
6054   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6055 }
6056 \NewDocumentCommand \@@_TopRule_ii: { o }
6057 {
6058   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6059   {
6060     \@@_hline:n
6061     {
6062       position = \int_eval:n { \c@iRow + 1 } ,
6063       tikz =
6064       {
6065         line-width = #1 ,
6066         yshift = 0.25 \arrayrulewidth ,
6067         shorten-< = - 0.5 \arrayrulewidth
6068       } ,
6069       total-width = #1
6070     }
6071   }
6072   \skip_vertical:n { \belowrulesep + #1 }
6073   \egroup
6074 }

```



```

6075 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6076 { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }
6077 \cs_new:Npn \@@_BottomRule_i:
6078 {
6079   \noalign \bgroup
6080   \peek_meaning:NTF [
6081     { \@@_BottomRule_ii: }
6082     { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6083   }
6084 \NewDocumentCommand \@@_BottomRule_ii: { o }
6085 {
6086   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6087   {
6088     \@@_hline:n
6089     {
6090       position = \int_eval:n { \c@iRow + 1 } ,
6091       tikz =
6092       {
6093         line-width = #1 ,
6094         yshift = 0.25 \arrayrulewidth ,
6095         shorten-< = - 0.5 \arrayrulewidth
6096       } ,
6097       total-width = #1 ,
6098     }
6099   }
6100   \skip_vertical:N \aboverulesep
6101   \@@_create_row_node_i:
6102   \skip_vertical:n { #1 }
6103   \egroup
6104 }
6105 \NewExpandableDocumentCommand { \@@_MidRule } { }
6106 { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }
6107 \cs_new:Npn \@@_MidRule_i:
6108 {
6109   \noalign \bgroup
6110   \peek_meaning:NTF [
6111     { \@@_MidRule_ii: }
6112     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6113   }
6114 \NewDocumentCommand \@@_MidRule_ii: { o }
6115 {
6116   \skip_vertical:N \aboverulesep
6117   \@@_create_row_node_i:
6118   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6119   {
6120     \@@_hline:n
6121     {
6122       position = \int_eval:n { \c@iRow + 1 } ,
6123       tikz =
6124       {
6125         line-width = #1 ,
6126         yshift = 0.25 \arrayrulewidth ,
6127         shorten-< = - 0.5 \arrayrulewidth
6128       } ,
6129       total-width = #1 ,
6130     }
6131   }
6132   \skip_vertical:n { \belowrulesep + #1 }
6133   \egroup
6134 }

```

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.

```
6135 \keys_define:nn { nicematrix / Rules }
6136 {
6137   position .int_set:N = \l_@@_position_int ,
6138   position .value_required:n = true ,
6139   start .int_set:N = \l_@@_start_int ,
6140   end .code:n =
6141     \bool_lazy_or:nnTF
6142     { \tl_if_empty_p:n { #1 } }
6143     { \str_if_eq_p:ee { #1 } { last } }
6144     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6145     { \int_set:Nn \l_@@_end_int { #1 } }
6146 }
```

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.

```
6147 \keys_define:nn { nicematrix / RulesBis }
6148 {
6149   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6150   multiplicity .initial:n = 1 ,
6151   dotted .bool_set:N = \l_@@_dotted_bool ,
6152   dotted .initial:n = false ,
6153   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]`.

```
6154   color .code:n =
6155     \@@_set_CT@arc@:n { #1 }
6156     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6157   color .value_required:n = true ,
6158   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6159   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.

```
6160   tikz .code:n =
6161     \IfPackageLoadedTF { tikz }
6162     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6163     { \@@_error:n { tikz-without-tikz } } ,
6164   tikz .value_required:n = true ,
6165   total-width .dim_set:N = \l_@@_rule_width_dim ,
6166   total-width .value_required:n = true ,
6167   width .meta:n = { total-width = #1 } ,
6168   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6169 }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
6170 \cs_new_protected:Npn \@@_vline:n #1
6171 {
```

The group is for the options.

```

6172   \group_begin:
6173   \int_set_eq:NN \l_@@_end_int \c@iRow
6174   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of |c|c|c| but only two columns used).

```

6175   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6176   \@@_vline_i:
6177   \group_end:
6178   }

6179   \cs_new_protected:Npn \@@_vline_i:
6180   {

```

`\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`.

```

6181   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6182   \int_step_variable:nNn \l_@@_start_int \l_@@_end_int
6183   \l_tmpa_tl
6184   {

```

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.

```

6185   \bool_gset_true:N \g_tmpa_bool

6186   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6187   { \@@_test_vline_in_block:nnnn #1 }
6188   \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6189   { \@@_test_vline_in_block:nnnn #1 }
6190   \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6191   { \@@_test_vline_in_stroken_block:nnn #1 }
6192   \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6193   \bool_if:NTF \g_tmpa_bool
6194   {
6195   \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.

```

6196   { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6197   }
6198   {
6199   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6200   {
6201   \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6202   \@@_vline_ii:
6203   \int_zero:N \l_@@_local_start_int
6204   }
6205   }
6206   }
6207   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6208   {
6209   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6210   \@@_vline_ii:
6211   }
6212   }

```

```

6213   \cs_new_protected:Npn \@@_test_in_corner_v:
6214   {
6215   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6216   {
6217   \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }

```

```

6218     { \bool_set_false:N \g_tmpa_bool }
6219   }
6220   {
6221     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6222     {
6223       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6224       { \bool_set_false:N \g_tmpa_bool }
6225       {
6226         \@@_if_in_corner:nT
6227         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6228         { \bool_set_false:N \g_tmpa_bool }
6229       }
6230     }
6231   }
6232 }

```

```

6233 \cs_new_protected:Npn \@@_vline_ii:
6234 {
6235   \tl_clear:N \l_@@_tikz_rule_tl
6236   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6237   \bool_if:NTF \l_@@_dotted_bool
6238     \@@_vline_iv:
6239     {
6240       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6241         \@@_vline_iii:
6242         \@@_vline_v:
6243     }
6244 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6245 \cs_new_protected:Npn \@@_vline_iii:
6246 {
6247   \pgfpicture
6248   \pgfrememberpicturepositiononpagetrue
6249   \pgf@relevantforpicturesizefalse
6250   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6251   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6252   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6253   \dim_set:Nn \l_tmpb_dim
6254     {
6255     \pgf@x
6256     - 0.5 \l_@@_rule_width_dim
6257     +
6258     ( \arrayrulewidth * \l_@@_multiplicity_int
6259       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6260     }
6261   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6262   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6263   \bool_lazy_all:nT
6264     {
6265     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6266     { \cs_if_exist_p:N \CT@drsc@ }
6267     { ! \tl_if_blank_p:o \CT@drsc@ }
6268     }
6269     {
6270     \group_begin:
6271     \CT@drsc@
6272     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6273     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6274     \dim_set:Nn \l_@@_tmpd_dim
6275       {
6276       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )

```

```

6277         * ( \l_@@_multiplicity_int - 1 )
6278     }
6279     \pgfpathrectanglecorners
6280     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6281     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6282     \pgfusepath { fill }
6283     \group_end:
6284 }
6285 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6286 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6287 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6288 {
6289     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6290     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6291     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6292     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6293 }
6294 \CT@arc@
6295 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6296 \pgfsetrectcap
6297 \pgfusepathqstroke
6298 \endpgfpicture
6299 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6300 \cs_new_protected:Npn \@@_vline_iv:
6301 {
6302     \pgfpicture
6303     \pgfrememberpicturepositiononpagetrue
6304     \pgf@relevantforpicturesizefalse
6305     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6306     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6307     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6308     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6309     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6310     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6311     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6312     \CT@arc@
6313     \@@_draw_line:
6314     \endpgfpicture
6315 }

```

The following code is for the case when the user uses the key `tikz`.

```

6316 \cs_new_protected:Npn \@@_vline_v:
6317 {
6318     \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).

```

6319     \CT@arc@
6320     \tl_if_empty:NF \l_@@_rule_color_tl
6321     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6322     \pgfrememberpicturepositiononpagetrue
6323     \pgf@relevantforpicturesizefalse
6324     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6325     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6326     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6327     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6328     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6329     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6330     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6331     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }

```

```

6332     ( \l_tmpb_dim , \l_tmpa_dim ) --
6333     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6334 \end { tikzpicture }
6335 }

```

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).

```

6336 \cs_new_protected:Npn \@@_draw_vlines:
6337 {
6338   \int_step_inline:nnn
6339     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6340     {
6341       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6342         \c@jCol
6343         { \int_eval:n { \c@jCol + 1 } }
6344     }
6345     {
6346       \str_if_eq:eeF \l_@@_vlines_clist { all }
6347       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6348       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6349     }
6350 }

```

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}`.

```

6351 \cs_new_protected:Npn \@@_hline:n #1
6352 {

```

The group is for the options.

```

6353   \group_begin:
6354   \int_zero_new:N \l_@@_end_int
6355   \int_set_eq:NN \l_@@_end_int \c@jCol
6356   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6357   \@@_hline_i:
6358   \group_end:
6359 }

6360 \cs_new_protected:Npn \@@_hline_i:
6361 {
6362   \int_zero_new:N \l_@@_local_start_int
6363   \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`.

```

6364   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6365   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6366     \l_tmpb_tl
6367     {

```

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.

```

6368     \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6369     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6370     { \@@_test_hline_in_block:nnnnn ##1 }

```

```

6371 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6372 { \@@_test_hline_in_block:nnnnn ##1 }
6373 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6374 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6375 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6376 \bool_if:NTF \g_tmpa_bool
6377 {
6378 \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.

```

6379 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6380 }
6381 {
6382 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6383 {
6384 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6385 \@@_hline_ii:
6386 \int_zero:N \l_@@_local_start_int
6387 }
6388 }
6389 }
6390 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6391 {
6392 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6393 \@@_hline_ii:
6394 }
6395 }

```

```

6396 \cs_new_protected:Npn \@@_test_in_corner_h:
6397 {
6398 \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6399 {
6400 \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6401 { \bool_set_false:N \g_tmpa_bool }
6402 }
6403 {
6404 \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6405 {
6406 \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6407 { \bool_set_false:N \g_tmpa_bool }
6408 {
6409 \@@_if_in_corner:nT
6410 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6411 { \bool_set_false:N \g_tmpa_bool }
6412 }
6413 }
6414 }
6415 }

```

```

6416 \cs_new_protected:Npn \@@_hline_ii:
6417 {
6418 \tl_clear:N \l_@@_tikz_rule_tl
6419 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6420 \bool_if:NTF \l_@@_dotted_bool
6421 \@@_hline_iv:
6422 {
6423 \tl_if_empty:NTF \l_@@_tikz_rule_tl
6424 \@@_hline_iii:
6425 \@@_hline_v:
6426 }
6427 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6428 \cs_new_protected:Npn \@@_hline_iii:
6429 {
6430   \pgfpicture
6431   \pgfrememberpicturepositiononpagetrue
6432   \pgf@relevantforpicturesizefalse
6433   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6434   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6435   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6436   \dim_set:Nn \l_tmpb_dim
6437     {
6438     \pgf@y
6439     - 0.5 \l_@@_rule_width_dim
6440     +
6441     ( \arrayrulewidth * \l_@@_multiplicity_int
6442     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6443     }
6444   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6445   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6446   \bool_lazy_all:nT
6447     {
6448     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6449     { \cs_if_exist_p:N \CT@drsc@ }
6450     { ! \tl_if_blank_p:o \CT@drsc@ }
6451     }
6452     {
6453     \group_begin:
6454     \CT@drsc@
6455     \dim_set:Nn \l_@@_tmpd_dim
6456       {
6457       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6458       * ( \l_@@_multiplicity_int - 1 )
6459       }
6460     \pgfpathrectanglecorners
6461       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6462       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6463     \pgfusepathqfill
6464     \group_end:
6465     }
6466   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6467   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6468   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6469     {
6470     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6471     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6472     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6473     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6474     }
6475   \CT@arc@
6476   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6477   \pgfsetrectcap
6478   \pgfusepathqstroke
6479   \endpgfpicture
6480 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).


```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6481 \cs_new_protected:Npn \@@_hline_iv:
6482 {
6483   \pgfpicture
6484   \pgfrememberpicturepositiononpagetrue
6485   \pgf@relevantforpicturesizefalse
6486   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6487   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6488   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6489   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6490   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6491   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6492   {
6493     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6494     \bool_if:NF \g_@@_delims_bool
6495     { \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.

```

6496     \tl_if_eq:NnF \g_@@_left_delim_tl (
6497       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6498     )
6499     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6500     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6501     \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6502     {
6503       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6504       \bool_if:NF \g_@@_delims_bool
6505       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6506       \tl_if_eq:NnF \g_@@_right_delim_tl )
6507       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6508     }
6509     \CT@arc@
6510     \@@_draw_line:
6511     \endpgfpicture
6512   }

```

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`).

```

6513 \cs_new_protected:Npn \@@_hline_v:
6514 {
6515   \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).

```

6516     \CT@arc@
6517     \tl_if_empty:NF \l_@@_rule_color_tl

```

```

6518     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6519 \pgfrememberpicturepositiononpagetrue
6520 \pgf@relevantforpicturesizefalse
6521 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6522 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6523 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6524 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6525 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6526 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6527 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6528 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6529   ( \l_tmpa_dim , \l_tmpb_dim ) --
6530   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6531 \end { tikzpicture }
6532 }

```

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).

```

6533 \cs_new_protected:Npn \@@_draw_hlines:
6534 {
6535   \int_step_inline:nnn
6536     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6537     {
6538       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6539         \c@iRow
6540         { \int_eval:n { \c@iRow + 1 } }
6541     }
6542     {
6543       \str_if_eq:eeF \l_@@_hlines_clist { all }
6544       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6545       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6546     }
6547 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6548 \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.

```

6549 \cs_set:Npn \@@_Hline_i:n #1
6550 {
6551   \peek_remove_spaces:n
6552     {
6553       \peek_meaning:NTF \Hline
6554       { \@@_Hline_ii:n { #1 + 1 } }
6555       { \@@_Hline_iii:n { #1 } }
6556     }
6557 }
6558 \cs_set:Npn \@@_Hline_ii:n #1 #2 { \@@_Hline_i:n { #1 } }
6559 \cs_set:Npn \@@_Hline_iii:n #1
6560 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6561 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6562 {
6563   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6564   \skip_vertical:N \l_@@_rule_width_dim
6565   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6566   {
6567     \@@_hline:n
6568     {
6569       multiplicity = #1 ,
6570       position = \int_eval:n { \c@iRow + 1 } ,

```

```

6571         total-width = \dim_use:N \l_@@_rule_width_dim ,
6572         #2
6573     }
6574 }
6575 \egroup
6576 }

```

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`).

```

6577 \cs_new_protected:Npn \@@_custom_line:n #1
6578 {
6579     \str_clear_new:N \l_@@_command_str
6580     \str_clear_new:N \l_@@_ccommand_str
6581     \str_clear_new:N \l_@@_letter_str
6582     \tl_clear_new:N \l_@@_other_keys_tl
6583     \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6584     \bool_lazy_all:nTF
6585     {
6586         { \str_if_empty_p:N \l_@@_letter_str }
6587         { \str_if_empty_p:N \l_@@_command_str }
6588         { \str_if_empty_p:N \l_@@_ccommand_str }
6589     }
6590     { \@@_error:n { No~letter~and~no~command } }
6591     { \@@_custom_line_i:o \l_@@_other_keys_tl }
6592 }
6593 \keys_define:nn { nicematrix / custom-line }
6594 {
6595     letter .str_set:N = \l_@@_letter_str ,
6596     letter .value_required:n = true ,
6597     command .str_set:N = \l_@@_command_str ,
6598     command .value_required:n = true ,
6599     ccommand .str_set:N = \l_@@_ccommand_str ,
6600     ccommand .value_required:n = true ,
6601 }

```

```

6602 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6603 \cs_new_protected:Npn \@@_custom_line_i:n #1
6604 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6605     \bool_set_false:N \l_@@_tikz_rule_bool
6606     \bool_set_false:N \l_@@_dotted_rule_bool
6607     \bool_set_false:N \l_@@_color_bool
6608     \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6609     \bool_if:NT \l_@@_tikz_rule_bool
6610     {
6611         \IfPackageLoadedF { tikz }
6612         { \@@_error:n { tikz-in-custom-line-without-tikz } }
6613         \bool_if:NT \l_@@_color_bool
6614         { \@@_error:n { color-in-custom-line-with-tikz } }
6615     }

```

```

6616 \bool_if:NT \l_@@_dotted_rule_bool
6617 {
6618   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6619   { \@@_error:n { key~multiplicity~with~dotted } }
6620 }
6621 \str_if_empty:NF \l_@@_letter_str
6622 {
6623   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6624   { \@@_error:n { Several~letters } }
6625   {
6626     \tl_if_in:NoTF
6627     \c_@@_forbidden_letters_str
6628     \l_@@_letter_str
6629     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6630   }

```

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.

```

6631       \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6632       { \@@_v_custom_line:n { #1 } }
6633     }
6634   }
6635 }
6636 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6637 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6638 }
6639 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6640 \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.

```

6641 \keys_define:nn { nicematrix / custom-line-bis }
6642 {
6643   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6644   multiplicity .initial:n = 1 ,
6645   multiplicity .value_required:n = true ,
6646   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6647   color .value_required:n = true ,
6648   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6649   tikz .value_required:n = true ,
6650   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6651   dotted .value_forbidden:n = true ,
6652   total-width .code:n = { } ,
6653   total-width .value_required:n = true ,
6654   width .code:n = { } ,
6655   width .value_required:n = true ,
6656   sep-color .code:n = { } ,
6657   sep-color .value_required:n = true ,
6658   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6659 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6660 \bool_new:N \l_@@_dotted_rule_bool
6661 \bool_new:N \l_@@_tikz_rule_bool
6662 \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`.

```

6663 \keys_define:nm { nicematrix / custom-line-width }
6664 {
6665   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6666   multiplicity .initial:n = 1 ,
6667   multiplicity .value_required:n = true ,
6668   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6669   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6670   \bool_set_true:N \l_@@_total_width_bool ,
6671   total-width .value_required:n = true ,
6672   width .meta:n = { total-width = #1 } ,
6673   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6674 }

```

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).

```

6675 \cs_new_protected:Npn \@@_h_custom_line:n #1
6676 {

```

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).

```

6677   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6678   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6679 }

```

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).

```

6680 \cs_new_protected:Npn \@@_c_custom_line:n #1
6681 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6682   \exp_args:Nc \NewExpandableDocumentCommand
6683   { nicematrix - \l_@@_ccommand_str }
6684   { 0 { } m }
6685   {
6686     \noalign
6687     {
6688       \@@_compute_rule_width:n { #1 , ##1 }
6689       \skip_vertical:n { \l_@@_rule_width_dim }
6690       \clist_map_inline:nn
6691       { ##2 }
6692       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6693     }
6694   }
6695   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6696 }

```

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*.

```

6697 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6698 {
6699   \tl_if_in:nnTF { #2 } { - }
6700   { \@@_cut_on_hyphen:w #2 \q_stop }
6701   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6702   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6703   {
6704     \@@_hline:n
6705     {
6706       #1 ,
6707       start = \l_tmpa_tl ,

```

```

6708         end = \l_tmpb_tl ,
6709         position = \int_eval:n { \c@iRow + 1 } ,
6710         total-width = \dim_use:N \l_@@_rule_width_dim
6711     }
6712 }
6713 }
6714 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6715 {
6716     \bool_set_false:N \l_@@_tikz_rule_bool
6717     \bool_set_false:N \l_@@_total_width_bool
6718     \bool_set_false:N \l_@@_dotted_rule_bool
6719     \keys_set_known:n { nicematrix / custom-line-width } { #1 }
6720     \bool_if:NF \l_@@_total_width_bool
6721     {
6722         \bool_if:NTF \l_@@_dotted_rule_bool
6723         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6724         {
6725             \bool_if:NF \l_@@_tikz_rule_bool
6726             {
6727                 \dim_set:Nn \l_@@_rule_width_dim
6728                 {
6729                     \arrayrulewidth * \l_@@_multiplicity_int
6730                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6731                 }
6732             }
6733         }
6734     }
6735 }
6736 \cs_new_protected:Npn \@@_v_custom_line:n #1
6737 {
6738     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6739     \tl_gput_right:Ne \g_@@_array_preamble_tl
6740     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6741     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6742     {
6743         \@@_vline:n
6744         {
6745             #1 ,
6746             position = \int_eval:n { \c@jCol + 1 } ,
6747             total-width = \dim_use:N \l_@@_rule_width_dim
6748         }
6749     }
6750     \@@_rec_preamble:n
6751 }
6752 \@@_custom_line:n
6753 { 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.

```

6754 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6755 {
6756     \int_compare:nNnT \l_tmpa_tl > { #1 }
6757     {
6758         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6759         {
6760             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }

```

```

6761         {
6762         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6763         { \bool_gset_false:N \g_tmpa_bool }
6764         }
6765     }
6766 }
6767 }

```

The same for vertical rules.

```

6768 \cs_new_protected:Npn \@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6769 {
6770   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6771   {
6772     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6773     {
6774       \int_compare:nNnT \l_tmpb_tl > { #2 }
6775       {
6776         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6777         { \bool_gset_false:N \g_tmpa_bool }
6778       }
6779     }
6780   }
6781 }

6782 \cs_new_protected:Npn \@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6783 {
6784   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6785   {
6786     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6787     {
6788       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6789       { \bool_gset_false:N \g_tmpa_bool }
6790       {
6791         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6792         { \bool_gset_false:N \g_tmpa_bool }
6793       }
6794     }
6795   }
6796 }

6797 \cs_new_protected:Npn \@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6798 {
6799   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6800   {
6801     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6802     {
6803       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6804       { \bool_gset_false:N \g_tmpa_bool }
6805       {
6806         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6807         { \bool_gset_false:N \g_tmpa_bool }
6808       }
6809     }
6810   }
6811 }

```

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.

```

6812 \cs_new_protected:Npn \@@_compute_corners:
6813 {
6814   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6815     { \@@_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`).

```

6816   \clist_clear:N \l_@@_corners_cells_clist
6817   \clist_map_inline:Nn \l_@@_corners_clist
6818     {
6819       \str_case:nnF { ##1 }
6820         {
6821           { NW }
6822           { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6823           { NE }
6824           { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6825           { SW }
6826           { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6827           { SE }
6828           { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6829         }
6830       { \@@_error:nn { bad~corner } { ##1 } }
6831     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6832   \clist_if_empty:NF \l_@@_corners_cells_clist
6833   {

```

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.

```

6834     \tl_gput_right:Ne \g_@@_aux_tl
6835     {
6836       \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6837         { \l_@@_corners_cells_clist }
6838     }
6839   }
6840 }

```

```

6841 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6842 {
6843   \int_step_inline:nnn { #1 } { #3 }
6844   {
6845     \int_step_inline:nnn { #2 } { #4 }
6846     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6847   }
6848 }

```

```

6849 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6850 {
6851   \cs_if_exist:cTF
6852     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6853     \prg_return_true:
6854     \prg_return_false:
6855 }

```

“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.

```
6856 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6857 {
```

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.

```
6858 \bool_set_false:N \l_tmpa_bool
6859 \int_zero_new:N \l_@@_last_empty_row_int
6860 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6861 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6862 {
6863   \bool_lazy_or:nnTF
6864   {
6865     \cs_if_exist_p:c
6866     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6867   }
6868   { \@@_if_in_block_p:nn { ##1 } { #2 } }
6869   { \bool_set_true:N \l_tmpa_bool }
6870   {
6871     \bool_if:NF \l_tmpa_bool
6872     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6873   }
6874 }
```

Now, you determine the last empty cell in the row of number 1.

```
6875 \bool_set_false:N \l_tmpa_bool
6876 \int_zero_new:N \l_@@_last_empty_column_int
6877 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6878 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6879 {
6880   \bool_lazy_or:nnTF
6881   {
6882     \cs_if_exist_p:c
6883     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6884   }
6885   { \@@_if_in_block_p:nn { #1 } { ##1 } }
6886   { \bool_set_true:N \l_tmpa_bool }
6887   {
6888     \bool_if:NF \l_tmpa_bool
6889     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6890   }
6891 }
```

Now, we loop over the rows.

```
6892 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6893 {
```

We treat the row number `##1` with another loop.

```
6894 \bool_set_false:N \l_tmpa_bool
6895 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6896 {
6897   \bool_lazy_or:nnTF
6898   { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6899   { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6900   { \bool_set_true:N \l_tmpa_bool }
6901   {
6902     \bool_if:NF \l_tmpa_bool
```

```

6903         {
6904             \int_set:Nn \l_@@_last_empty_column_int { ##### }
6905             \clist_put_right:Nn
6906                 \l_@@_corners_cells_clist
6907                 { ##1 - ##### }
6908             \cs_set_nopar:cpn { @@ _ corner _ ##1 - ##### } { }
6909         }
6910     }
6911 }
6912 }
6913 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6914 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6915 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6916 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6917 \keys_define:nn { nicematrix / NiceMatrixBlock }
6918 {
6919     auto-columns-width .code:n =
6920     {
6921         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6922         \dim_gzero_new:N \g_@@_max_cell_width_dim
6923         \bool_set_true:N \l_@@_auto_columns_width_bool
6924     }
6925 }

6926 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6927 {
6928     \int_gincr:N \g_@@_NiceMatrixBlock_int
6929     \dim_zero:N \l_@@_columns_width_dim
6930     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6931     \bool_if:NT \l_@@_block_auto_columns_width_bool
6932     {
6933         \cs_if_exist:cT
6934             { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6935         {
6936             \dim_set:Nn \l_@@_columns_width_dim
6937             {
6938                 \use:c
6939                     { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6940             }
6941         }
6942     }
6943 }

```

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`).

```
6944 {
6945   \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.

```
6946   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6947   {
6948     \bool_if:NT \l_@@_block_auto_columns_width_bool
6949     {
6950       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6951       \iow_shipout:Ne \@mainaux
6952       {
6953         \cs_gset:cpn
6954         { @@ _ 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.

```
6955         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6956       }
6957       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6958     }
6959   }
6960   \ignorespacesafterend
6961 }
```

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).

```
6962 \cs_new_protected:Npn \@@_create_extra_nodes:
6963 {
6964   \bool_if:nTF \l_@@_medium_nodes_bool
6965   {
6966     \bool_if:NTF \l_@@_no_cell_nodes_bool
6967     { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6968     {
6969       \bool_if:NTF \l_@@_large_nodes_bool
6970       \@@_create_medium_and_large_nodes:
6971       \@@_create_medium_nodes:
6972     }
6973   }
6974   {
6975     \bool_if:NT \l_@@_large_nodes_bool
6976     {
6977       \bool_if:NTF \l_@@_no_cell_nodes_bool
6978       { \@@_error:n { extra-nodes-with-no-cell-nodes } }
6979       \@@_create_large_nodes:
6980     }
6981   }
6982 }
```

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`.

```

6983 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6984 {
6985   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6986   {
6987     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6988     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6989     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6990     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6991   }
6992   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6993   {
6994     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6995     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6996     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6997     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6998   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6999   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7000   {
7001     \int_step_variable:nnNn
7002     \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.

```

7003     {
7004       \cs_if_exist:cT
7005       { 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`.

```

7006     {
7007       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
7008       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
7009       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
7010       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7011       {
7012         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
7013         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
7014       }

```

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`.

```

7015       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
7016       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
7017       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
7018       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
7019       {
7020         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

7021             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
7022         }
7023     }
7024 }
7025 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7026 \int_step_variable:nNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7027 {
7028     \dim_compare:nNnT
7029     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7030     {
7031         \@@_qpoint:n { row - \@@_i: - base }
7032         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7033         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7034     }
7035 }
7036 \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7037 {
7038     \dim_compare:nNnT
7039     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7040     {
7041         \@@_qpoint:n { col - \@@_j: }
7042         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7043         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7044     }
7045 }
7046 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7047 \cs_new_protected:Npn \@@_create_medium_nodes:
7048 {
7049     \pgfpicture
7050     \pgfrememberpicturepositiononpagetrue
7051     \pgf@relevantforpicturesizefalse
7052     \@@_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”.

```

7053     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7054     \@@_create_nodes:
7055     \endpgfpicture
7056 }

```

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:`.

```

7057 \cs_new_protected:Npn \@@_create_large_nodes:
7058 {
7059     \pgfpicture
7060     \pgfrememberpicturepositiononpagetrue
7061     \pgf@relevantforpicturesizefalse
7062     \@@_computations_for_medium_nodes:
7063     \@@_computations_for_large_nodes:
7064     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7065     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7066 \endpgfpicture
7067 }
7068 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7069 {
7070 \pgfpicture
7071 \pgfrememberpicturepositiononpagetrue
7072 \pgf@relevantforpicturesizefalse
7073 \@@_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”.

```

7074 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
7075 \@@_create_nodes:
7076 \@@_computations_for_large_nodes:
7077 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7078 \@@_create_nodes:
7079 \endpgfpicture
7080 }

```

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.

```

7081 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7082 {
7083 \int_set_eq:NN \l_@@_first_row_int \c_one_int
7084 \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`.

```

7085 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7086 {
7087 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7088 {
7089 (
7090 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7091 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7092 )
7093 / 2
7094 }
7095 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7096 { l_@@_row _ \@@_i: _ min _ dim }
7097 }
7098 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7099 {
7100 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7101 {
7102 (
7103 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7104 \dim_use:c
7105 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7106 )
7107 / 2
7108 }
7109 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7110 { l_@@_column _ \@@_j: _ max _ dim }
7111 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7112 \dim_sub:cn
7113 { l_@@_column _ 1 _ min _ dim }
7114 \l_@@_left_margin_dim
7115 \dim_add:cn
7116 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7117 \l_@@_right_margin_dim
7118 }

```

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`).

```

7119 \cs_new_protected:Npn \@@_create_nodes:
7120 {
7121   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7122   {
7123     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7124     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7125     \@@_pgf_rect_node:nnnnn
7126     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7127     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7128     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7129     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7130     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7131     \str_if_empty:NF \l_@@_name_str
7132     {
7133       \pgfnodealias
7134       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7135       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7136     }
7137   }
7138 }

```

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 .

```

7139   \seq_map_pairwise_function:NNN
7140   \g_@@_multicolumn_cells_seq
7141   \g_@@_multicolumn_sizes_seq
7142   \@@_node_for_multicolumn:nn
7143 }

7144 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7145 {
7146   \cs_set_nopar:Npn \@@_i: { #1 }
7147   \cs_set_nopar:Npn \@@_j: { #2 }
7148 }

```

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”).

```

7149 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7150 {
7151   \@@_extract_coords_values: #1 \q_stop
7152   \@@_pgf_rect_node:nnnnn
7153   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7154   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7155   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7156   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
7157   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7158   \str_if_empty:NF \l_@@_name_str
7159   {
7160     \pgfnodealias
7161     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7162     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7163   }
7164 }

```

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).

```

7165 \keys_define:nm { nicematrix / Block / FirstPass }
7166 {
7167   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7168             \bool_set_true:N \l_@@_p_block_bool ,
7169   j .value_forbidden:n = true ,
7170   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7171   l .value_forbidden:n = true ,
7172   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7173   r .value_forbidden:n = true ,
7174   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7175   c .value_forbidden:n = true ,
7176   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7177   L .value_forbidden:n = true ,
7178   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7179   R .value_forbidden:n = true ,
7180   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7181   C .value_forbidden:n = true ,
7182   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7183   t .value_forbidden:n = true ,
7184   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7185   T .value_forbidden:n = true ,
7186   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7187   b .value_forbidden:n = true ,
7188   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7189   B .value_forbidden:n = true ,
7190   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7191   m .value_forbidden:n = true ,
7192   v-center .meta:n = m ,
7193   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7194   p .value_forbidden:n = true ,
7195   color .code:n =
7196     \@@_color:n { #1 }
7197     \tl_set_rescan:Nnn
7198       \l_@@_draw_tl
7199       { \char_set_catcode_other:N ! }
7200       { #1 } ,
7201   color .value_required:n = true ,
7202   respect-arraystretch .code:n =
7203     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7204   respect-arraystretch .value_forbidden:n = true ,
7205 }

```

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.

```

7206 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7207 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7208 {

```

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).

```

7209   \peek_remove_spaces:n

```



```

7210 {
7211   \tl_if_blank:nTF { #2 }
7212     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7213     {
7214       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7215       \@@_Block_i_czech \@@_Block_i
7216       #2 \q_stop
7217     }
7218   { #1 } { #3 } { #4 }
7219 }
7220 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7221 \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`.

```

7222 {
7223   \char_set_catcode_active:N -
7224   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7225 }

```

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.

```

7226 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7227 {

```

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).

```

7228   \bool_lazy_or:nnTF
7229     { \tl_if_blank_p:n { #1 } }
7230     { \str_if_eq_p:ee { * } { #1 } }
7231     { \int_set:Nn \l_tmpa_int { 100 } }
7232     { \int_set:Nn \l_tmpa_int { #1 } }
7233   \bool_lazy_or:nnTF
7234     { \tl_if_blank_p:n { #2 } }
7235     { \str_if_eq_p:ee { * } { #2 } }
7236     { \int_set:Nn \l_tmpb_int { 100 } }
7237     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7238   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7239     {
7240       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7241         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7242         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7243     }
7244     { \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.

```

7245   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }

```

```

7246   \tl_set:Nc \l_tmpa_tl
7247   {
7248     { \int_use:N \c@iRow }
7249     { \int_use:N \c@jCol }
7250     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7251     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7252   }

```

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).

```

7253   \bool_set_false:N \l_tmpa_bool
7254   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7255   { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7256   \bool_case:nF
7257   {
7258     \l_tmpa_bool                { \@@_Block_vii:eennn }
7259     \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.

```

7260   \l_@@_X_bool                { \@@_Block_v:eennn }
7261   { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7262   { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7263   { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7264   }
7265   { \@@_Block_v:eennn }
7266   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7267   }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don’t use the key `p`. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.

`#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of `key=values` pairs, `#4` are the tokens to put before the potential math mode and before the composition of the block and `#5` is the label (=content) of the block.

```

7268   \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7269   \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7270   {
7271     \int_gincr:N \g_@@_block_box_int
7272     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7273     {
7274       \tl_gput_right:Ne \g_@@_pre_code_after_tl
7275       {
7276         \@@_actually_diagbox:nnnnnn
7277         { \int_use:N \c@iRow }
7278         { \int_use:N \c@jCol }
7279         { \int_eval:n { \c@iRow + #1 - 1 } }
7280         { \int_eval:n { \c@jCol + #2 - 1 } }
7281         { \g_@@_row_style_tl \exp_not:n { ##1 } }
7282         { \g_@@_row_style_tl \exp_not:n { ##2 } }

```

```

7283     }
7284   }
7285   \box_gclear_new:c
7286   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7287   \hbox_gset:cn
7288   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7289   {

```

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}`).

```

7290     \tl_if_empty:NTF \l_@@_color_tl
7291     { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7292     { \@@_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`.

```

7293     \int_compare:nNnT { #1 } = \c_one_int
7294     {
7295       \int_if_zero:nTF \c@iRow
7296       {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7297     \cs_set_eq:NN \Block \@@_NullBlock:
7298     \l_@@_code_for_first_row_tl
7299   }
7300   {
7301     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7302     {
7303       \cs_set_eq:NN \Block \@@_NullBlock:
7304       \l_@@_code_for_last_row_tl
7305     }
7306   }
7307   \g_@@_row_style_tl
7308 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```
7309     \@@_reset_arraystretch:
7310     \dim_zero:N \extrarowheight
```

`#4` is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```
7311     #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.).

```
7312     \@@_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}`.

```
7313     \bool_if:NTF \l_@@_tabular_bool
7314     {
7315         \bool_lazy_all:nTF
7316         {
7317             { \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`.

```
7318         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7319         { ! \g_@@_rotate_bool }
7320     }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7321     {
7322         \use:e
7323         {
```

The `\exp_not:N` is mandatory before `\begin`.

```
7324         \exp_not:N \begin { minipage }%
7325         [ \str_lowercase:o \l_@@_vpos_block_str ]
7326         { \l_@@_col_width_dim }
7327         \str_case:on \l_@@_hpos_block_str
7328         { c \centering r \raggedleft l \raggedright }
7329     }
7330     #5
7331     \end { minipage }
7332 }
```

In the other cases, we use a `{tabular}`.

```
7333     {
7334         \bool_if:NT \c_@@_testphase_table_bool
7335         { \tagpdfsetup { table / tagging = presentation } }
7336         \use:e
7337         {
7338             \exp_not:N \begin { tabular }%
7339             [ \str_lowercase:o \l_@@_vpos_block_str ]
7340             { @ { } \l_@@_hpos_block_str @ { } }
7341         }
7342         #5
7343         \end { tabular }
7344     }
7345 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7346     {
7347         \c_math_toggle_token
7348         \use:e
7349         {
```

```

7350         \exp_not:N \begin { array }%
7351         [ \str_lowercase:o \l_@@_vpos_block_str ]
7352         { @ { } \l_@@_hpos_block_str @ { } }
7353     }
7354     #5
7355     \end { array }
7356     \c_math_toggle_token
7357 }
7358 }

```

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).

```

7359     \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.

```

7360     \int_compare:nNnT { #2 } = \c_one_int
7361     {
7362         \dim_gset:Nn \g_@@_blocks_wd_dim
7363         {
7364             \dim_max:nn
7365             \g_@@_blocks_wd_dim
7366             {
7367                 \box_wd:c
7368                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7369             }
7370         }
7371     }

```

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.

```

7372     \bool_lazy_and:nnT
7373     { \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.

```

7374     { \str_if_empty_p:N \l_@@_vpos_block_str }
7375     {
7376         \dim_gset:Nn \g_@@_blocks_ht_dim
7377         {
7378             \dim_max:nn
7379             \g_@@_blocks_ht_dim
7380             {
7381                 \box_ht:c
7382                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7383             }
7384         }
7385         \dim_gset:Nn \g_@@_blocks_dp_dim
7386         {
7387             \dim_max:nn
7388             \g_@@_blocks_dp_dim
7389             {
7390                 \box_dp:c
7391                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7392             }
7393         }
7394     }
7395     \seq_gput_right:Ne \g_@@_blocks_seq
7396     {
7397         \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.

```

7398     {
7399         \exp_not:n { #3 } ,
7400         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7401         \bool_if:NT \g_@@_rotate_bool
7402         {
7403             \bool_if:NTF \g_@@_rotate_c_bool
7404             { m }
7405             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7406         }
7407     }
7408     {
7409         \box_use_drop:c
7410         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7411     }
7412 }
7413 \bool_set_false:N \g_@@_rotate_c_bool
7414 }

```

```

7415 \cs_new:Npn \@@_adjust_hpos_rotate:
7416 {
7417     \bool_if:NT \g_@@_rotate_bool
7418     {
7419         \str_set:Ne \l_@@_hpos_block_str
7420         {
7421             \bool_if:NTF \g_@@_rotate_c_bool
7422             { c }
7423             {
7424                 \str_case:onF \l_@@_vpos_block_str
7425                 { b l B l t r T r }
7426                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7427             }
7428         }
7429     }
7430 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7431 \cs_new_protected:Npn \@@_rotate_box_of_block:
7432 {
7433     \box_grotate:cn
7434     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7435     { 90 }
7436     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7437     {
7438         \vbox_gset_top:cn
7439         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7440         {
7441             \skip_vertical:n { 0.8 ex }
7442             \box_use:c
7443             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7444         }
7445     }
7446     \bool_if:NT \g_@@_rotate_c_bool
7447     {
7448         \hbox_gset:cn
7449         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7450     {
7451     \c_math_toggle_token
7452     \vcenter
7453     {
7454     \box_use:c
7455     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7456     }
7457     \c_math_toggle_token
7458     }
7459 }
7460 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7461 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7462 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7463 {
7464 \seq_gput_right:Ne \g_@@_blocks_seq
7465 {
7466 \l_tmpa_tl
7467 { \exp_not:n { #3 } }
7468 {
7469 \bool_if:NTF \l_@@_tabular_bool
7470 {
7471 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7472 \@@_reset_arraystretch:
7473 \exp_not:n
7474 {
7475 \dim_zero:N \extrarowheight
7476 #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).

```

7477 \bool_if:NT \c_@@_testphase_table_bool
7478 { \tag_stop:n { table } }
7479 \use:e
7480 {
7481 \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7482 { @ { } \l_@@_hpos_block_str @ { } }
7483 }
7484 #5
7485 \end { tabular }
7486 }
7487 \group_end:
7488 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7489 {
7490 \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7491 \@@_reset_arraystretch:
7492 \exp_not:n
7493 {

```

```

7494         \dim_zero:N \extrarowheight
7495         #4
7496         \c_math_toggle_token
7497         \use:e
7498         {
7499             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7500             { @ { } \l_@@_hpos_block_str @ { } }
7501         }
7502         #5
7503         \end { array }
7504         \c_math_toggle_token
7505     }
7506     \group_end:
7507 }
7508 }
7509 }
7510 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7511 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7512 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7513 {
7514     \seq_gput_right:Ne \g_@@_blocks_seq
7515     {
7516         \l_tmpa_tl
7517         { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7518         { { \exp_not:n { #4 #5 } } }
7519     }
7520 }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7521 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7522 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7523 {
7524     \seq_gput_right:Ne \g_@@_blocks_seq
7525     {
7526         \l_tmpa_tl
7527         { \exp_not:n { #3 } }
7528         { \exp_not:n { #4 #5 } }
7529     }
7530 }

```

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).

```

7531 \keys_define:nn { nicematrix / Block / SecondPass }
7532 {
7533     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7534     ampersand-in-blocks .default:n = true ,
7535     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7536     tikz .code:n =
7537         \IfPackageLoadedTF { tikz }
7538             { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7539             { \@@_error:n { tikz-key-without-tikz } } ,
7540     tikz .value_required:n = true ,
7541     fill .code:n =
7542         \tl_set_rescan:Nnn

```



```

7543     \l_@@_fill_tl
7544     { \char_set_catcode_other:N ! }
7545     { #1 } ,
7546 fill .value_required:n = true ,
7547 opacity .tl_set:N = \l_@@_opacity_tl ,
7548 opacity .value_required:n = true ,
7549 draw .code:n =
7550     \tl_set_rescan:Nnn
7551     \l_@@_draw_tl
7552     { \char_set_catcode_other:N ! }
7553     { #1 } ,
7554 draw .default:n = default ,
7555 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7556 rounded-corners .default:n = 4 pt ,
7557 color .code:n =
7558     \@@_color:n { #1 }
7559     \tl_set_rescan:Nnn
7560     \l_@@_draw_tl
7561     { \char_set_catcode_other:N ! }
7562     { #1 } ,
7563 borders .clist_set:N = \l_@@_borders_clist ,
7564 borders .value_required:n = true ,
7565 hvlines .meta:n = { vlines , hlines } ,
7566 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7567 vlines .default:n = true ,
7568 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7569 hlines .default:n = true ,
7570 line-width .dim_set:N = \l_@@_line_width_dim ,
7571 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7572 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7573     \bool_set_true:N \l_@@_p_block_bool ,
7574 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7575 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7576 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7577 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7578     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7579 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7580     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7581 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7582     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7583 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7584 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7585 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7586 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7587 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7588 m .value_forbidden:n = true ,
7589 v-center .meta:n = m ,
7590 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7591 p .value_forbidden:n = true ,
7592 name .tl_set:N = \l_@@_block_name_str ,
7593 name .value_required:n = true ,
7594 name .initial:n = ,
7595 respect-arraystretch .code:n =
7596     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7597 respect-arraystretch .value_forbidden:n = true ,
7598 transparent .bool_set:N = \l_@@_transparent_bool ,
7599 transparent .default:n = true ,
7600 transparent .initial:n = false ,
7601 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7602 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construc-

tion of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7603 \cs_new_protected:Npn \@@_draw_blocks:
7604 {
7605   \bool_if:NTF \c_@@_recent_array_bool
7606     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7607     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7608   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7609 }
7610 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7611 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7612   \int_zero_new:N \l_@@_last_row_int
7613   \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format i - j . However, the user is allowed to omit i or j (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7614   \int_compare:nNnTF { #3 } > { 98 }
7615     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7616     { \int_set:Nn \l_@@_last_row_int { #3 } }
7617   \int_compare:nNnTF { #4 } > { 98 }
7618     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7619     { \int_set:Nn \l_@@_last_col_int { #4 } }
7620   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7621     {
7622       \bool_lazy_and:nnTF
7623         \l_@@_preamble_bool
7624         {
7625           \int_compare_p:n
7626             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7627         }
7628         {
7629           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7630           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7631           \@@_msg_redirect_name:nn { columns-not-used } { none }
7632         }
7633         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7634     }
7635   {
7636     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7637       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7638       {
7639         \@@_Block_v:nneenn
7640           { #1 }
7641           { #2 }
7642           { \int_use:N \l_@@_last_row_int }
7643           { \int_use:N \l_@@_last_col_int }
7644           { #5 }
7645           { #6 }
7646       }
7647     }
7648 }

```

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

```

7649 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7650 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7651 {

```

The group is for the keys.

```

7652 \group_begin:
7653 \int_compare:nNt { #1 } = { #3 }
7654 { \str_set:Nn \l_@@_vpos_block_str { t } }
7655 \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`.

```

7656 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7657 \bool_lazy_and:nnT
7658 \l_@@_vlines_block_bool
7659 { ! \l_@@_ampersand_bool }
7660 {
7661 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7662 {
7663 \@@_vlines_block:nnn
7664 { \exp_not:n { #5 } }
7665 { #1 - #2 }
7666 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7667 }
7668 }
7669 \bool_if:NT \l_@@_hlines_block_bool
7670 {
7671 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7672 {
7673 \@@_hlines_block:nnn
7674 { \exp_not:n { #5 } }
7675 { #1 - #2 }
7676 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7677 }
7678 }
7679 \bool_if:NF \l_@@_transparent_bool
7680 {
7681 \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7682 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7683 \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7684 { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7685 }
7686 }

```

```

7687 \tl_if_empty:NF \l_@@_draw_tl
7688 {
7689 \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7690 { \@@_error:n { hlines~with~color } }
7691 \tl_gput_right:Ne \g_nicematrix_code_after_tl
7692 {
7693 \@@_stroke_block:nnn

```

#5 are the options

```

7694 { \exp_not:n { #5 } }
7695 { #1 - #2 }
7696 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7697 }
7698 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7699 { { #1 } { #2 } { #3 } { #4 } }
7700 }

```

```

7701 \clist_if_empty:NF \l_@@_borders_clist
7702 {
7703   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7704   {
7705     \@@_stroke_borders_block:nnn
7706     { \exp_not:n { #5 } }
7707     { #1 - #2 }
7708     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7709   }
7710 }
7711 \tl_if_empty:NF \l_@@_fill_tl
7712 {
7713   \@@_add_opacity_to_fill:
7714   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7715   {
7716     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7717     { #1 - #2 }
7718     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7719     { \dim_use:N \l_@@_rounded_corners_dim }
7720   }
7721 }
7722 \seq_if_empty:NF \l_@@_tikz_seq
7723 {
7724   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7725   {
7726     \@@_block_tikz:nnnnn
7727     { \seq_use:Nn \l_@@_tikz_seq { , } }
7728     { #1 }
7729     { #2 }
7730     { \int_use:N \l_@@_last_row_int }
7731     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7732   }
7733 }
7734 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7735 {
7736   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7737   {
7738     \@@_actually_diagbox:nnnnnn
7739     { #1 }
7740     { #2 }
7741     { \int_use:N \l_@@_last_row_int }
7742     { \int_use:N \l_@@_last_col_int }
7743     { \exp_not:n { ##1 } }
7744     { \exp_not:n { ##2 } }
7745   }
7746 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node 1-1-block

our block		one
three	four	two
six	seven	five
		eight

We highlight the node 1-1-block-short

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7747 \pgfpicture
7748 \pgfrememberpicturepositiononpagetrue
7749 \pgf@relevantforpicturesizefalse
7750 \@@_qpoint:n { row - #1 }
7751 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7752 \@@_qpoint:n { col - #2 }
7753 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7754 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7755 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7756 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7757 \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.

```

7758 \@@_pgf_rect_node:nnnnn
7759 { \@@_env: - #1 - #2 - block }
7760 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7761 \str_if_empty:NF \l_@@_block_name_str
7762 {
7763 \pgfnodealias
7764 { \@@_env: - \l_@@_block_name_str }
7765 { \@@_env: - #1 - #2 - block }
7766 \str_if_empty:NF \l_@@_name_str
7767 {
7768 \pgfnodealias
7769 { \l_@@_name_str - \l_@@_block_name_str }
7770 { \@@_env: - #1 - #2 - block }
7771 }
7772 }

```

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.

```

7773 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7774 {
7775 \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.

```

7776 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7777 {

```

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.

```

7778 \cs_if_exist:cT
7779 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7780 {
7781 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7782 {
7783 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7784 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7785 }
7786 }
7787 }

```

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.

```

7788     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7789     {
7790       \@@_qpoint:n { col - #2 }
7791       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7792     }
7793     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7794     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7795     {
7796       \cs_if_exist:cT
7797       { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7798       {
7799         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7800         {
7801           \pgfpointanchor
7802           { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7803           { east }
7804           \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7805         }
7806       }
7807     }
7808     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7809     {
7810       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7811       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7812     }
7813     \@@_pgf_rect_node:nnnnn
7814     { \@@_env: - #1 - #2 - block - short }
7815     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7816   }

```

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.

```

7817     \bool_if:NT \l_@@_medium_nodes_bool
7818     {
7819       \@@_pgf_rect_node:nnn
7820       { \@@_env: - #1 - #2 - block - medium }
7821       { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7822       {
7823         \pgfpointanchor
7824         { \@@_env:
7825           - \int_use:N \l_@@_last_row_int
7826           - \int_use:N \l_@@_last_col_int - medium
7827         }
7828         { south-east }
7829       }
7830     }
7831     \endpgfpicture

7832     \bool_if:NTF \l_@@_ampersand_bool
7833     {
7834       \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7835       \int_zero_new:N \l_@@_split_int
7836       \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7837       \pgfpicture
7838       \pgfrememberpicturepositiononpagetrue
7839       \pgf@relevantforpicturesizefalse
7840
7841       \@@_qpoint:n { row - #1 }
7842       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7843       \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }

```

```

7844 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7845 \@@_qpoint:n { col - #2 }
7846 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7847 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7848 \dim_set:Nn \l_tmpb_dim
7849 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7850 \bool_lazy_or:nnT
7851 \l_@@_vlines_block_bool
7852 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7853 {
7854 \int_step_inline:nn { \l_@@_split_int - 1 }
7855 {
7856 \pgfpathmoveto
7857 {
7858 \pgfpoint
7859 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7860 \l_@@_tmpc_dim
7861 }
7862 \pgfpathlineto
7863 {
7864 \pgfpoint
7865 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7866 \l_@@_tmpd_dim
7867 }
7868 \CT@arc@
7869 \pgfsetlinewidth { 1.1 \arrayrulewidth }
7870 \pgfsetrectcap
7871 \pgfusepathqstroke
7872 }
7873 }
7874 \@@_qpoint:n { row - #1 - base }
7875 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7876 \int_step_inline:nn \l_@@_split_int
7877 {
7878 \group_begin:
7879 \dim_set:Nn \col@sep
7880 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7881 \pgftransformshift
7882 {
7883 \pgfpoint
7884 {
7885 \l_tmpa_dim + ##1 \l_tmpb_dim -
7886 \str_case:on \l_@@_hpos_block_str
7887 {
7888 l { \l_tmpb_dim + \col@sep }
7889 c { 0.5 \l_tmpb_dim }
7890 r { \col@sep }
7891 }
7892 }
7893 { \l_@@_tmpc_dim }
7894 }
7895 \pgfset { inner~sep = \c_zero_dim }
7896 \pgfnode
7897 { rectangle }
7898 {
7899 \str_case:on \l_@@_hpos_block_str
7900 {
7901 c { base }
7902 l { base~west }
7903 r { base~east }
7904 }
7905 }
7906 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }

```

```

7907         \group_end:
7908     }
7909     \endpgfpicture
7910 }

```

Now the case where there is no ampersand & in the content of the block.

```

7911 {
7912     \bool_if:NTF \l_@@_p_block_bool
7913     {

```

When the final user has used the key p, we have to compute the width.

```

7914         \pgfpicture
7915         \pgfrememberpicturepositiononpagetrue
7916         \pgf@relevantforpicturesizefalse
7917         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7918         {
7919             \@@_qpoint:n { col - #2 }
7920             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7921             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7922         }
7923         {
7924             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7925             \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7926             \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7927         }
7928         \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7929     \endpgfpicture
7930     \hbox_set:Nn \l_@@_cell_box
7931     {
7932         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7933         { \g_tmpb_dim }
7934         \str_case:on \l_@@_hpos_block_str
7935         { c \centering r \raggedleft l \raggedright j { } }
7936         #6
7937         \end { minipage }
7938     }
7939 }
7940 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7941 \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.

```

7942     \pgfpicture
7943     \pgfrememberpicturepositiononpagetrue
7944     \pgf@relevantforpicturesizefalse
7945     \bool_lazy_any:nTF
7946     {
7947         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7948         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7949         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7950         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7951     }
7952     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7953         \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.

```

7954         \bool_if:nT \g_@@_last_col_found_bool
7955         {
7956             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7957             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7958         }

```


`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7959     \tl_set:Ne \l_tmpa_tl
7960     {
7961         \str_case:on \l_@@_vpos_block_str
7962         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7963         { } { % added 2024-06-29
7964             \str_case:on \l_@@_hpos_block_str
7965             {
7966                 c { center }
7967                 l { west }
7968                 r { east }
7969                 j { center }
7970             }
7971         }
7972     c {
7973         \str_case:on \l_@@_hpos_block_str
7974         {
7975             c { center }
7976             l { west }
7977             r { east }
7978             j { center }
7979         }
7980     }
7981     T {
7982         \str_case:on \l_@@_hpos_block_str
7983         {
7984             c { north }
7985             l { north-west }
7986             r { north-east }
7987             j { north }
7988         }
7989     }
7990     B {
7991         \str_case:on \l_@@_hpos_block_str
7992         {
7993             c { south }
7994             l { south-west }
7995             r { south-east }
7996             j { south }
7997         }
7998     }
7999     }
8000     }
8001     }
8002     }
8003     }
8004     \pgftransformshift
8005     {
8006         \pgfpointanchor
8007         {
8008             \@@_env: - #1 - #2 - block
8009             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8010         }
8011         { \l_tmpa_tl }
8012     }
8013     \pgfset { inner~sep = \c_zero_dim }
8014     \pgfnode
8015     { rectangle }
8016     { \l_tmpa_tl }
8017     { \box_use_drop:N \l_@@_cell_box } { } { }

```

```
8018     }
```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```
8019     {
8020         \pgfextracty \l_tmpa_dim
8021         {
8022             \@@_qpoint:n
8023             {
8024                 row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8025                 - base
8026             }
8027         }
8028         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```
8029         \pgfpointanchor
8030         {
8031             \@@_env: - #1 - #2 - block
8032             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8033         }
8034         {
8035             \str_case:on \l_@@_hpos_block_str
8036             {
8037                 c { center }
8038                 l { west }
8039                 r { east }
8040                 j { center }
8041             }
8042         }
```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```
8043         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8044         \pgfset { inner-sep = \c_zero_dim }
8045         \pgfnode
8046         { rectangle }
8047         {
8048             \str_case:on \l_@@_hpos_block_str
8049             {
8050                 c { base }
8051                 l { base~west }
8052                 r { base~east }
8053                 j { base }
8054             }
8055         }
8056         { \box_use_drop:N \l_@@_cell_box } { } { }
8057     }
8058     \endpgfpicture
8059 }
8060 \group_end:
8061 }
```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```
8062 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8063 {
8064     \pgfpicture
8065     \pgfrememberpicturepositiononpagetrue
8066     \pgf@relevantforpicturesizefalse
8067     \pgfpathrectanglecorners
8068     { \pgfpoint { #2 } { #3 } }
8069     { \pgfpoint { #4 } { #5 } }
8070     \pgfsetfillcolor { #1 }
8071     \pgfusepath { fill }
```

```

8072 \endpgfpicture
8073 }

```

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).

```

8074 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8075 {
8076   \tl_if_empty:NF \l_@@_opacity_tl
8077   {
8078     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8079     {
8080       \tl_set:Ne \l_@@_fill_tl
8081       {
8082         [ opacity = \l_@@_opacity_tl ,
8083         \tl_tail:o \l_@@_fill_tl
8084       }
8085     }
8086     {
8087       \tl_set:Ne \l_@@_fill_tl
8088       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8089     }
8090   }
8091 }

```

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).

```

8092 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8093 {
8094   \group_begin:
8095   \tl_clear:N \l_@@_draw_tl
8096   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8097   \keys_set_known:n { nicematrix / BlockStroke } { #1 }
8098   \pgfpicture
8099   \pgfrememberpicturepositiononpagetrue
8100   \pgf@relevantforpicturesizefalse
8101   \tl_if_empty:NF \l_@@_draw_tl
8102   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8103     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8104     { \CT@arc@ }
8105     { \@@_color:o \l_@@_draw_tl }
8106   }
8107   \pgfsetcornersarced
8108   {
8109     \pgfpoint
8110     { \l_@@_rounded_corners_dim }
8111     { \l_@@_rounded_corners_dim }
8112   }
8113   \@@_cut_on_hyphen:w #2 \q_stop
8114   \int_compare:nNnF \l_tmpa_tl > \c@iRow
8115   {
8116     \int_compare:nNnF \l_tmpb_tl > \c@jCol
8117     {
8118       \@@_qpoint:n { row - \l_tmpa_tl }
8119       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8120       \@@_qpoint:n { col - \l_tmpb_tl }
8121       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8122       \@@_cut_on_hyphen:w #3 \q_stop

```

```

8123 \int_compare:nNnT \l_tmpa_tl > \c@iRow
8124   { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8125 \int_compare:nNnT \l_tmpb_tl > \c@jCol
8126   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8127 @@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8128 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8129 @@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8130 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8131 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8132 \pgfpathrectanglecorners
8133   { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8134   { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8135 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8136   { \pgfusepathqstroke }
8137   { \pgfusepath { stroke } }
8138 }
8139 }
8140 \endpgfpicture
8141 \group_end:
8142 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8143 \keys_define:nn { nicematrix / BlockStroke }
8144 {
8145   color .tl_set:N = \l_@@_draw_tl ,
8146   draw .code:n =
8147     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8148   draw .default:n = default ,
8149   line-width .dim_set:N = \l_@@_line_width_dim ,
8150   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8151   rounded-corners .default:n = 4 pt
8152 }

```

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).

```

8153 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8154 {
8155   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8156   \keys_set_known:n { nicematrix / BlockBorders } { #1 }
8157   @@_cut_on_hyphen:w #2 \q_stop
8158   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8159   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8160   @@_cut_on_hyphen:w #3 \q_stop
8161   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8162   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8163   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8164     {
8165       \use:e
8166       {
8167         \@@_vline:n
8168         {
8169           position = ##1 ,
8170           start = \l_@@_tmpc_tl ,
8171           end = \int_eval:n { \l_tmpa_tl - 1 } ,
8172           total-width = \dim_use:N \l_@@_line_width_dim
8173         }
8174       }
8175     }
8176 }
8177 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8178 {
8179   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth

```

```

8180 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8181 \@@_cut_on_hyphen:w #2 \q_stop
8182 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8183 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8184 \@@_cut_on_hyphen:w #3 \q_stop
8185 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8186 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8187 \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8188 {
8189   \use:e
8190   {
8191     \@@_hline:n
8192     {
8193       position = ##1 ,
8194       start = \l_@@_tmpd_tl ,
8195       end = \int_eval:n { \l_tmpb_tl - 1 } ,
8196       total-width = \dim_use:N \l_@@_line_width_dim
8197     }
8198   }
8199 }
8200 }

```

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).

```

8201 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8202 {
8203   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8204   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8205   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8206     { \@@_error:n { borders~forbidden } }
8207     {
8208       \tl_clear_new:N \l_@@_borders_tikz_tl
8209       \keys_set:no
8210         { nicematrix / OnlyForTikzInBorders }
8211         \l_@@_borders_clist
8212         \@@_cut_on_hyphen:w #2 \q_stop
8213         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8214         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8215         \@@_cut_on_hyphen:w #3 \q_stop
8216         \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8217         \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8218         \@@_stroke_borders_block_i:
8219       }
8220   }
8221 \hook_gput_code:nnn { begindocument } { . }
8222 {
8223   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8224     {
8225       \c_@@_pgfortikzpicture_tl
8226       \@@_stroke_borders_block_ii:
8227       \c_@@_endpgfortikzpicture_tl
8228     }
8229 }
8230 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8231 {
8232   \pgfrememberpicturepositiononpagetrue
8233   \pgf@relevantforpicturesizefalse
8234   \CT@arc@
8235   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8236   \clist_if_in:NnT \l_@@_borders_clist { right }
8237     { \@@_stroke_vertical:n \l_tmpb_tl }

```

```

8238 \clist_if_in:NnT \l_@@_borders_clist { left }
8239   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8240 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8241   { \@@_stroke_horizontal:n \l_tmpa_tl }
8242 \clist_if_in:NnT \l_@@_borders_clist { top }
8243   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8244 }
8245 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8246 {
8247   tikz .code:n =
8248     \cs_if_exist:NTF \tikzpicture
8249       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8250       { \@@_error:n { tikz~in~borders~without~tikz } } } ,
8251   tikz .value_required:n = true ,
8252   top .code:n = ,
8253   bottom .code:n = ,
8254   left .code:n = ,
8255   right .code:n = ,
8256   unknown .code:n = \@@_error:n { bad~border }
8257 }

```

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).

```

8258 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8259 {
8260   \@@_qpoint:n \l_@@_tmpc_tl
8261   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8262   \@@_qpoint:n \l_tmpa_tl
8263   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8264   \@@_qpoint:n { #1 }
8265   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8266     {
8267       \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8268       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8269       \pgfusepathqstroke
8270     }
8271     {
8272       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8273         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8274     }
8275 }

```

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).

```

8276 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8277 {
8278   \@@_qpoint:n \l_@@_tmpd_tl
8279   \clist_if_in:NnTF \l_@@_borders_clist { left }
8280     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8281     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8282   \@@_qpoint:n \l_tmpb_tl
8283   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8284   \@@_qpoint:n { #1 }
8285   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8286     {
8287       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8288       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8289       \pgfusepathqstroke
8290     }
8291     {
8292       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8293         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8294     }

```

```
8295 }
```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```
8296 \keys_define:nn { nicematrix / BlockBorders }
8297 {
8298   borders .clist_set:N = \l_@@_borders_clist ,
8299   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8300   rounded-corners .default:n = 4 pt ,
8301   line-width .dim_set:N = \l_@@_line_width_dim
8302 }
```

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.

```
8303 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8304 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8305 {
8306   \begin { tikzpicture }
8307   \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```
8308   \clist_map_inline:nn { #1 }
8309   {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8310     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8311     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8312     (
8313     [
8314       xshift = \dim_use:N \l_@@_offset_dim ,
8315       yshift = - \dim_use:N \l_@@_offset_dim
8316     ]
8317     #2 -| #3
8318     )
8319     rectangle
8320     (
8321     [
8322       xshift = - \dim_use:N \l_@@_offset_dim ,
8323       yshift = \dim_use:N \l_@@_offset_dim
8324     ]
8325     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8326     ) ;
8327   }
8328   \end { tikzpicture }
8329 }
```

```
8330 \keys_define:nn { nicematrix / SpecialOffset }
8331 { 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.

```
8332 \cs_new_protected:Npn \@@_NullBlock:
8333 { \@@_collect_options:n { \@@_NullBlock_i: } }
8334 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8335 { }
```

27 How to draw the dotted lines transparently

```

8336 \cs_set_protected:Npn \@@_renew_matrix:
8337 {
8338   \RenewDocumentEnvironment { pmatrix } { }
8339     { \pNiceMatrix }
8340     { \endpNiceMatrix }
8341   \RenewDocumentEnvironment { vmatrix } { }
8342     { \vNiceMatrix }
8343     { \endvNiceMatrix }
8344   \RenewDocumentEnvironment { Vmatrix } { }
8345     { \VNiceMatrix }
8346     { \endVNiceMatrix }
8347   \RenewDocumentEnvironment { bmatrix } { }
8348     { \bNiceMatrix }
8349     { \endbNiceMatrix }
8350   \RenewDocumentEnvironment { Bmatrix } { }
8351     { \BNiceMatrix }
8352     { \endBNiceMatrix }
8353 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8354 \keys_define:nn { nicematrix / Auto }
8355 {
8356   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8357   columns-type .value_required:n = true ,
8358   l .meta:n = { columns-type = l } ,
8359   r .meta:n = { columns-type = r } ,
8360   c .meta:n = { columns-type = c } ,
8361   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8362   delimiters / color .value_required:n = true ,
8363   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8364   delimiters / max-width .default:n = true ,
8365   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8366   delimiters .value_required:n = true ,
8367   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8368   rounded-corners .default:n = 4 pt
8369 }
8370 \NewDocumentCommand \AutoNiceMatrixWithDelims
8371 { m m 0 { } } > { \SplitArgument { 1 } { - } } m 0 { } m ! 0 { } }
8372 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8373 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8374 {

```

The group is for the protection of the keys.

```

8375   \group_begin:
8376   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8377   \use:e
8378   {
8379     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8380       { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8381       [ \exp_not:o \l_tmpa_tl ]
8382   }
8383   \int_if_zero:nT \l_@@_first_row_int
8384   {
8385     \int_if_zero:nT \l_@@_first_col_int { & }
8386     \prg_replicate:nn { #4 - 1 } { & }
8387     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\

```



```

8388     }
8389     \prg_replicate:nn { #3 }
8390     {
8391         \int_if_zero:nT \l_@@_first_col_int { & }
8392         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8393         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\\
8394     }
8395     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8396     {
8397         \int_if_zero:nT \l_@@_first_col_int { & }
8398         \prg_replicate:nn { #4 - 1 } { & }
8399         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\\
8400     }
8401     \end { NiceArrayWithDelims }
8402     \group_end:
8403 }

8404 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8405 {
8406     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8407     {
8408         \bool_gset_true:N \g_@@_delims_bool
8409         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8410         \AutoNiceMatrixWithDelims { #2 } { #3 }
8411     }
8412 }

8413 \@@_define_com:nnn p ( )
8414 \@@_define_com:nnn b [ ]
8415 \@@_define_com:nnn v | |
8416 \@@_define_com:nnn V \| \|
8417 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8418 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8419 {
8420     \group_begin:
8421     \bool_gset_false:N \g_@@_delims_bool
8422     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8423     \group_end:
8424 }

```

29 The redefinition of the command `\dotfill`

```

8425 \cs_set_eq:NN \@@_old_dotfill \dotfill
8426 \cs_new_protected:Npn \@@_dotfill:
8427 {

```

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}`).

```

8428     \@@_old_dotfill
8429     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8430 }

```

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`.

```

8431 \cs_new_protected:Npn \@@_dotfill_i:
8432 { \dim_compare:nNt { \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.

```

8433 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8434 {
8435   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8436   {
8437     \@@_actually_diagbox:nnnnnn
8438     { \int_use:N \c@iRow }
8439     { \int_use:N \c@jCol }
8440     { \int_use:N \c@iRow }
8441     { \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.

```

8442     { \g_@@_row_style_tl \exp_not:n { #1 } }
8443     { \g_@@_row_style_tl \exp_not:n { #2 } }
8444   }

```

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.

```

8445   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8446   {
8447     { \int_use:N \c@iRow }
8448     { \int_use:N \c@jCol }
8449     { \int_use:N \c@iRow }
8450     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8451     { }
8452   }
8453 }

```

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.

```

8454 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8455 {
8456   \pgfpicture
8457   \pgf@relevantforpicturesizefalse
8458   \pgfrememberpicturepositiononpagetrue
8459   \@@_qpoint:n { row - #1 }
8460   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8461   \@@_qpoint:n { col - #2 }
8462   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8463   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8464   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8465   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8466   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8467   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8468   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8469   {

```

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.

```

8470     \CT@arc@
8471     \pgfsetroundcap
8472     \pgfusepathqstroke
8473   }
8474   \pgfset { inner~sep = 1 pt }
8475   \pgfscope
8476   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8477   \pgfnode { rectangle } { south~west }
8478     {
8479     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8480     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8481     \end { minipage }
8482   }
8483   { }
8484   { }
8485 \endpgfscope
8486 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8487 \pgfnode { rectangle } { north~east }
8488   {
8489     \begin { minipage } { 20 cm }
8490     \raggedleft
8491     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8492     \end { minipage }
8493   }
8494   { }
8495   { }
8496 \endpgfpicture
8497 }

```

31 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 83.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8498 \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 `\`.

```

8499 \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`.

```

8500 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8501   {
8502     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8503     \@@_CodeAfter_iv:n
8504   }

```

We catch the argument of the command `\end` (in #1).

```

8505 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8506   {

```

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.

```
8507 \str_if_eq:eeTF \@currenvir { #1 }
8508 { \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`.

```
8509 {
8510 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8511 \@@_CodeAfter_ii:n
8512 }
8513 }
```

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.

```
8514 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8515 {
8516 \pgfpicture
8517 \pgfrememberpicturepositiononpagetrue
8518 \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.

```
8519 \@@_qpoint:n { row - 1 }
8520 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8521 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8522 \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).

```
8523 \bool_if:nTF { #3 }
8524 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8525 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8526 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8527 {
8528 \cs_if_exist:cT
8529 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8530 {
8531 \pgfpointanchor
8532 { \@@_env: - ##1 - #2 }
8533 { \bool_if:nTF { #3 } { west } { east } }
8534 \dim_set:Nn \l_tmpa_dim
8535 { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8536 }
8537 }
```

Now we can put the delimiter with a node of PGF.

```

8538 \pgfset { inner~sep = \c_zero_dim }
8539 \dim_zero:N \nulldelimiterspace
8540 \pgftransformshift
8541 {
8542   \pgfpoint
8543   { \l_tmpa_dim }
8544   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8545 }
8546 \pgfnode
8547 { rectangle }
8548 { \bool_if:nTF { #3 } { east } { west } }
8549 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8550 \nullfont
8551 \c_math_toggle_token
8552 \@@_color:o \l_@@_delimiters_color_tl
8553 \bool_if:nTF { #3 } { \left #1 } { \left . }
8554 \vcenter
8555 {
8556   \nullfont
8557   \hrule \@height
8558   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8559   \@depth \c_zero_dim
8560   \@width \c_zero_dim
8561 }
8562 \bool_if:nTF { #3 } { \right . } { \right #1 }
8563 \c_math_toggle_token
8564 }
8565 { }
8566 { }
8567 \endpgfpicture
8568 }

```

33 The command `\SubMatrix`

```

8569 \keys_define:nn { nicematrix / sub-matrix }
8570 {
8571   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8572   extra-height .value_required:n = true ,
8573   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8574   left-xshift .value_required:n = true ,
8575   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8576   right-xshift .value_required:n = true ,
8577   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8578   xshift .value_required:n = true ,
8579   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8580   delimiters / color .value_required:n = true ,
8581   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8582   slim .default:n = true ,
8583   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8584   hlines .default:n = all ,
8585   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8586   vlines .default:n = all ,
8587   hvlines .meta:n = { hlines, vlines } ,
8588   hvlines .value_forbidden:n = true
8589 }
8590 \keys_define:nn { nicematrix }
8591 {
8592   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

8593 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8594 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8595 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8596 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8597 \keys_define:nm { nicematrix / SubMatrix }
8598 {
8599   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8600   delimiters / color .value_required:n = true ,
8601   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8602   hlines .default:n = all ,
8603   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8604   vlines .default:n = all ,
8605   hvlines .meta:n = { hlines, vlines } ,
8606   hvlines .value_forbidden:n = true ,
8607   name .code:n =
8608     \tl_if_empty:nTF { #1 }
8609     { \@@_error:n { Invalid-name } }
8610     {
8611       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8612       {
8613         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8614         { \@@_error:nm { Duplicate-name-for-SubMatrix } { #1 } }
8615         {
8616           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8617           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8618         }
8619       }
8620       { \@@_error:n { Invalid-name } }
8621     } ,
8622   name .value_required:n = true ,
8623   rules .code:n = \keys_set:nm { nicematrix / rules } { #1 } ,
8624   rules .value_required:n = true ,
8625   code .tl_set:N = \l_@@_code_tl ,
8626   code .value_required:n = true ,
8627   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8628 }

8629 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8630 {
8631   \peek_remove_spaces:n
8632   {
8633     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8634     {
8635       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8636       [
8637         delimiters / color = \l_@@_delimiters_color_tl ,
8638         hlines = \l_@@_submatrix_hlines_clist ,
8639         vlines = \l_@@_submatrix_vlines_clist ,
8640         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8641         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8642         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8643         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8644         #5
8645       ]
8646     }
8647     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8648   }
8649 }

8650 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8651 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }

```

```

8652 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8653 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8654 {
8655   \seq_gput_right:Ne \g_@@_submatrix_seq
8656   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8657     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8658     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8659     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8660     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8661   }
8662 }

```

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`.

```

8663 \hook_gput_code:nnn { begindocument } { . }
8664 {
8665   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8666   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8667   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8668   {
8669     \peek_remove_spaces:n
8670     {
8671       \@@_sub_matrix:nnnnnnn
8672       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8673     }
8674   }
8675 }

```

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).

```

8676 \NewDocumentCommand \@@_compute_i_j:nn
8677 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8678 { \@@_compute_i_j:nnnn #1 #2 }
8679 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8680 {
8681   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8682   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8683   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8684   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8685   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8686   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8687   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8688   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8689   \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

8690     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8691 \tl_if_eq:NnT \l_@@_last_j_tl { last }
8692     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8693 }

8694 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8695 {
8696   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8697   \@@_compute_i_j:nn { #2 } { #3 }
8698   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8699     { \cs_set_nopar:Npn \arraystretch { 1 } }
8700   \bool_lazy_or:nnTF
8701     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8702     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8703     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8704   {
8705     \str_clear_new:N \l_@@_submatrix_name_str
8706     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8707     \pgfpicture
8708     \pgfrememberpicturepositiononpagetrue
8709     \pgf@relevantforpicturesizefalse
8710     \pgfset { inner~sep = \c_zero_dim }
8711     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8712     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currying.

```

8713   \bool_if:NTF \l_@@_submatrix_slim_bool
8714     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8715     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8716   {
8717     \cs_if_exist:cT
8718       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8719     {
8720       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8721       \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8722         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8723     }
8724     \cs_if_exist:cT
8725       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8726     {
8727       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8728       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8729         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8730     }
8731   }
8732   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8733     { \@@_error:nn { Impossible~delimiter } { left } }
8734   {
8735     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8736       { \@@_error:nn { Impossible~delimiter } { right } }
8737     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8738   }
8739   \endpgfpicture
8740 }
8741 \group_end:
8742 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8743 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8744 {
8745   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8746   \dim_set:Nn \l_@@_y_initial_dim

```



```

8747 {
8748   \fp_to_dim:n
8749   {
8750     \pgf@y
8751     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8752   }
8753 }
8754 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8755 \dim_set:Nn \l_@@_y_final_dim
8756 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8757 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8758 {
8759   \cs_if_exist:cT
8760   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8761   {
8762     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8763     \dim_set:Nn \l_@@_y_initial_dim
8764     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8765   }
8766   \cs_if_exist:cT
8767   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8768   {
8769     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8770     \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8771     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8772   }
8773 }
8774 \dim_set:Nn \l_tmpa_dim
8775 {
8776   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8777   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8778 }
8779 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8780 \group_begin:
8781 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8782 \@@_set_CT@arc@o \l_@@_rules_color_tl
8783 \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`.

```

8784 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8785 {
8786   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8787   {
8788     \int_compare:nNnT
8789     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8790     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8791     \@@_qpoint:n { col - ##1 }
8792     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8793     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8794     \pgfusepathqstroke
8795   }
8796 }
8797 }

```

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.

```

8798 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }

```

```

8799 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8800 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8801 {
8802   \bool_lazy_and:nnTF
8803     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8804     {
8805       \int_compare_p:nNn
8806         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8807     {
8808       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8809       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8810       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8811       \pgfusepathqstroke
8812     }
8813     { \@@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8814 }

```

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.

```

8815 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8816 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8817 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8818 {
8819   \bool_lazy_and:nnTF
8820     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8821     {
8822       \int_compare_p:nNn
8823         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8824     {
8825       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8826   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8827   \dim_set:Nn \l_tmpa_dim
8828     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8829   \str_case:nn { #1 }
8830     {
8831       ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8832       [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8833       \} { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8834     }
8835   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8836   \dim_set:Nn \l_tmpb_dim
8837     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8838   \str_case:nn { #2 }
8839     {
8840       ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8841       ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8842       \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8843     }
8844   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8845   \pgfusepathqstroke
8846   \group_end:
8847 }
8848 { \@@_error:nmn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8849 }

```

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).

```

8850   \str_if_empty:NF \l_@@_submatrix_name_str

```

```

8851     {
8852     \l_@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8853     \l_@@_x_initial_dim \l_@@_y_initial_dim
8854     \l_@@_x_final_dim \l_@@_y_final_dim
8855     }
8856 \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`.

```

8857 \begin { pgfscope }
8858 \pgftransformshift
8859 {
8860 \pgfpoint
8861 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8862 { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8863 }
8864 \str_if_empty:NTF \l_@@_submatrix_name_str
8865 { \l_@@_node_left:nn #1 { } }
8866 { \l_@@_node_left:nn #1 { \l_@@_env: - \l_@@_submatrix_name_str - left } }
8867 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8868 \pgftransformshift
8869 {
8870 \pgfpoint
8871 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8872 { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8873 }
8874 \str_if_empty:NTF \l_@@_submatrix_name_str
8875 { \l_@@_node_right:nnnn #2 { } { #3 } { #4 } }
8876 {
8877 \l_@@_node_right:nnnn #2
8878 { \l_@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8879 }
8880 \cs_set_eq:NN \pgfpointanchor \l_@@_pgfpointanchor:n
8881 \flag_clear_new:N \l_@@_code_flag
8882 \l_@@_code_tl
8883 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $row-i$, $col-j$ and $i-lj$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8884 \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.

```

8885 \cs_new_protected:Npn \l_@@_pgfpointanchor:n #1
8886 {
8887 \use:e
8888 { \exp_not:N \l_@@_old_pgfpointanchor { \l_@@_pgfpointanchor_i:nn #1 } }
8889 }

```

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 -.

```

8890 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8891 { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8892 \tl_const:Nn \c_@@_integers_alist_tl
8893 {
8894   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8895   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8896   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8897   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8898 }

```

```

8899 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8900 {

```

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`.

```

8901   \tl_if_empty:nTF { #2 }
8902   {
8903     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8904     {
8905       \flag_raise:N \l_@@_code_flag
8906       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8907       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8908       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8909     }
8910     { #1 }
8911   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8912     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8913   }

```

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`).

```

8914 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8915 {
8916   \str_case:nnF { #1 }
8917   {
8918     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8919     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8920   }

```

Now the case of a node of the form $i-j$.

```

8921   {
8922     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8923     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8924   }
8925 }

```

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`).

```

8926 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8927 {
8928   \pgfnode
8929   { rectangle }

```

```

8930 { east }
8931 {
8932   \nullfont
8933   \c_math_toggle_token
8934   \@@_color:o \l_@@_delimiters_color_tl
8935   \left #1
8936   \vcenter
8937   {
8938     \nullfont
8939     \hrule \@height \l_tmpa_dim
8940             \@depth \c_zero_dim
8941             \@width \c_zero_dim
8942   }
8943   \right .
8944   \c_math_toggle_token
8945 }
8946 { #2 }
8947 { }
8948 }

```

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.

```

8949 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8950 {
8951   \pgfnode
8952   { rectangle }
8953   { west }
8954   {
8955     \nullfont
8956     \c_math_toggle_token
8957     \colorlet { current-color } { . }
8958     \@@_color:o \l_@@_delimiters_color_tl
8959     \left .
8960     \vcenter
8961     {
8962       \nullfont
8963       \hrule \@height \l_tmpa_dim
8964               \@depth \c_zero_dim
8965               \@width \c_zero_dim
8966     }
8967     \right #1
8968     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8969     ^ { \color { current-color } \smash { #4 } }
8970     \c_math_toggle_token
8971   }
8972   { #2 }
8973   { }
8974 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8975 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8976 {
8977   \peek_remove_spaces:n
8978   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8979 }

```

```

8980 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8981 {
8982   \peek_remove_spaces:n
8983   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8984 }

8985 \keys_define:nn { nicematrix / Brace }
8986 {
8987   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8988   left-shorten .default:n = true ,
8989   left-shorten .value_forbidden:n = true ,
8990   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8991   right-shorten .default:n = true ,
8992   right-shorten .value_forbidden:n = true ,
8993   shorten .meta:n = { left-shorten , right-shorten } ,
8994   shorten .value_forbidden:n = true ,
8995   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8996   yshift .value_required:n = true ,
8997   yshift .initial:n = \c_zero_dim ,
8998   color .tl_set:N = \l_tmpa_tl ,
8999   color .value_required:n = true ,
9000   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
9001 }

```

#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*.

```

9002 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
9003 {
9004   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

9005   \@@_compute_i_j:nn { #1 } { #2 }
9006   \bool_lazy_or:nnTF
9007     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
9008     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
9009     {
9010       \str_if_eq:eeTF { #5 } { under }
9011       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
9012       { \@@_error:nn { Construct-too-large } { \OverBrace } }
9013     }
9014   {
9015     \tl_clear:N \l_tmpa_tl
9016     \keys_set:nn { nicematrix / Brace } { #4 }
9017     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
9018     \pgfpicture
9019     \pgfrememberpicturepositiononpagetrue
9020     \pgf@relevantforpicturesizefalse
9021     \bool_if:NT \l_@@_brace_left_shorten_bool
9022     {
9023       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9024       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9025         {
9026           \cs_if_exist:cT
9027             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9028             {
9029               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9030             }
9031           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9032             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9033         }
9034     }
9035   }

```

```

9036 \bool_lazy_or:nnT
9037   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9038   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9039   {
9040     \@@_qpoint:n { col - \l_@@_first_j_tl }
9041     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9042   }
9043 \bool_if:NT \l_@@_brace_right_shorten_bool
9044   {
9045     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9046     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9047     {
9048       \cs_if_exist:cT
9049         { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9050         {
9051           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9052           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9053           { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
9054         }
9055     }
9056   }
9057 \bool_lazy_or:nnT
9058   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9059   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9060   {
9061     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9062     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9063   }
9064 \pgfset { inner~sep = \c_zero_dim }
9065 \str_if_eq:eeTF { #5 } { under }
9066   { \@@_underbrace_i:n { #3 } }
9067   { \@@_overbrace_i:n { #3 } }
9068 \endpgfpicture
9069 }
9070 \group_end:
9071 }

```

The argument is the text to put above the brace.

```

9072 \cs_new_protected:Npn \@@_overbrace_i:n #1
9073   {
9074     \@@_qpoint:n { row - \l_@@_first_i_tl }
9075     \pgftransformshift
9076     {
9077       \pgfpoint
9078       { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9079       { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9080     }
9081 \pgfnode
9082   { rectangle }
9083   { south }
9084   {
9085     \vtop
9086     {
9087       \group_begin:
9088       \everycr { }
9089       \halign
9090       {
9091         \hfil ## \hfil \crcr
9092         \bool_if:NTF \l_@@_tabular_bool
9093           { \begin { tabular } { c } #1 \end { tabular } }
9094           { $ \begin { array } { c } #1 \end { array } $ }
9095         \cr
9096         \c_math_toggle_token
9097         \overbrace

```

```

9098         {
9099             \hbox_to_wd:nn
9100             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9101             { }
9102         }
9103         \c_math_toggle_token
9104         \cr
9105     }
9106     \group_end:
9107 }
9108 }
9109 { }
9110 { }
9111 }

```

The argument is the text to put under the brace.

```

9112 \cs_new_protected:Npn \@@_underbrace_i:n #1
9113 {
9114     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9115     \pgftransformshift
9116     {
9117         \pgfpoint
9118         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9119         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9120     }
9121     \pgfnode
9122     { rectangle }
9123     { north }
9124     {
9125         \group_begin:
9126         \everycr { }
9127         \vbox
9128         {
9129             \halign
9130             {
9131                 \hfil ## \hfil \crcr
9132                 \c_math_toggle_token
9133                 \underbrace
9134                 {
9135                     \hbox_to_wd:nn
9136                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9137                     { }
9138                 }
9139                 \c_math_toggle_token
9140                 \cr
9141                 \bool_if:NTF \l_@@_tabular_bool
9142                 { \begin { tabular } { c } #1 \end { tabular } }
9143                 { $ \begin { array } { c } #1 \end { array } $ }
9144                 \cr
9145             }
9146         }
9147         \group_end:
9148     }
9149     { }
9150     { }
9151 }

```

35 The command TikzEveryCell


```

9152 \bool_new:N \l_@@_not_empty_bool
9153 \bool_new:N \l_@@_empty_bool
9154
9155 \keys_define:nn { nicematrix / TikzEveryCell }
9156 {
9157   not-empty .code:n =
9158     \bool_lazy_or:nnTF
9159       \l_@@_in_code_after_bool
9160       \g_@@_recreate_cell_nodes_bool
9161       { \bool_set_true:N \l_@@_not_empty_bool }
9162       { \@@_error:n { detection~of~empty~cells } } ,
9163   not-empty .value_forbidden:n = true ,
9164   empty .code:n =
9165     \bool_lazy_or:nnTF
9166       \l_@@_in_code_after_bool
9167       \g_@@_recreate_cell_nodes_bool
9168       { \bool_set_true:N \l_@@_empty_bool }
9169       { \@@_error:n { detection~of~empty~cells } } ,
9170   empty .value_forbidden:n = true ,
9171   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9172 }
9173
9174
9175 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9176 {
9177   \IfPackageLoadedTF { tikz }
9178   {
9179     \group_begin:
9180     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9181     \tl_set:Nn \l_tmpa_tl { { #2 } }
9182     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9183       { \@@_for_a_block:nnnnn ##1 }
9184     \@@_all_the_cells:
9185     \group_end:
9186   }
9187   { \@@_error:n { TikzEveryCell~without~tikz } }
9188 }
9189
9190 \tl_new:N \@@_i_tl
9191 \tl_new:N \@@_j_tl
9192
9193
9194 \cs_new_protected:Nn \@@_all_the_cells:
9195 {
9196   \int_step_variable:nNn \c@iRow \@@_i_tl
9197   {
9198     \int_step_variable:nNn \c@jCol \@@_j_tl
9199     {
9200       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9201       {
9202         \clist_if_in:NeF \l_@@_corners_cells_clist
9203           { \@@_i_tl - \@@_j_tl }
9204         {
9205           \bool_set_false:N \l_tmpa_bool
9206           \cs_if_exist:cTF
9207             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9208             {
9209               \bool_if:NF \l_@@_empty_bool
9210               { \bool_set_true:N \l_tmpa_bool }
9211             }
9212         }

```

```

9213             \bool_if:NF \l_@@_not_empty_bool
9214             { \bool_set_true:N \l_tmpa_bool }
9215         }
9216     \bool_if:NT \l_tmpa_bool
9217     {
9218         \@@_block_tikz:onnnn
9219         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9220     }
9221 }
9222 }
9223 }
9224 }
9225 }
9226
9227 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9228 {
9229     \bool_if:NF \l_@@_empty_bool
9230     {
9231         \@@_block_tikz:onnnn
9232         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9233     }
9234     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9235 }
9236
9237 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9238 {
9239     \int_step_inline:nnn { #1 } { #3 }
9240     {
9241         \int_step_inline:nnn { #2 } { #4 }
9242         { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9243     }
9244 }

```

36 The command `\ShowCellNames`

```

9245 \NewDocumentCommand \@@_ShowCellNames { }
9246 {
9247     \bool_if:NT \l_@@_in_code_after_bool
9248     {
9249         \pgfpicture
9250         \pgfrememberpicturepositiononpagetrue
9251         \pgf@relevantforpicturesizefalse
9252         \pgfpathrectanglecorners
9253         { \@@_qpoint:n { 1 } }
9254         {
9255             \@@_qpoint:n
9256             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9257         }
9258         \pgfsetfillopacity { 0.75 }
9259         \pgfsetfillcolor { white }
9260         \pgfusepathqfill
9261         \endpgfpicture
9262     }
9263     \dim_gzero_new:N \g_@@_tmpc_dim
9264     \dim_gzero_new:N \g_@@_tmpd_dim
9265     \dim_gzero_new:N \g_@@_tmpe_dim
9266     \int_step_inline:nn \c@iRow
9267     {
9268         \bool_if:NTF \l_@@_in_code_after_bool
9269         {
9270             \pgfpicture
9271             \pgfrememberpicturepositiononpagetrue

```

```

9272     \pgf@relevantforpicturesizefalse
9273   }
9274   { \begin { pgfpicture } }
9275   \@@_qpoint:n { row - ##1 }
9276   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9277   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9278   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9279   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9280   \bool_if:NTF \l_@@_in_code_after_bool
9281     { \endpgfpicture }
9282     { \end { pgfpicture } }
9283   \int_step_inline:nn \c@jCol
9284   {
9285     \hbox_set:Nn \l_tmpa_box
9286     {
9287       \normalfont \Large \sffamily \bfseries
9288       \bool_if:NTF \l_@@_in_code_after_bool
9289         { \color { red } }
9290         { \color { red ! 50 } }
9291       ##1 - ####1
9292     }
9293     \bool_if:NTF \l_@@_in_code_after_bool
9294     {
9295       \pgfpicture
9296       \pgfrememberpicturerepositiononpagetrue
9297       \pgf@relevantforpicturesizefalse
9298     }
9299     { \begin { pgfpicture } }
9300     \@@_qpoint:n { col - ####1 }
9301     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9302     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9303     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9304     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9305     \bool_if:NTF \l_@@_in_code_after_bool
9306       { \endpgfpicture }
9307       { \end { pgfpicture } }
9308     \fp_set:Nn \l_tmpa_fp
9309     {
9310       \fp_min:nn
9311       {
9312         \fp_min:nn
9313         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9314         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9315       }
9316       { 1.0 }
9317     }
9318     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9319     \pgfpicture
9320     \pgfrememberpicturerepositiononpagetrue
9321     \pgf@relevantforpicturesizefalse
9322     \pgftransformshift
9323     {
9324       \pgfpoint
9325       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9326       { \dim_use:N \g_tmpa_dim }
9327     }
9328     \pgfnode
9329     { rectangle }
9330     { center }
9331     { \box_use:N \l_tmpa_box }
9332     { }
9333     { }
9334     \endpgfpicture

```

```

9335     }
9336   }
9337 }

```

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.

```

9338 \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.

```

9339 \bool_new:N \g_@@_footnote_bool

9340 \msg_new:nmmm { nicematrix } { Unknown~key~for~package }
9341 {
9342   The~key~'\l_keys_key_str'~is~unknown. \\
9343   That~key~will~be~ignored. \\
9344   For~a~list~of~the~available~keys,~type~H~<return>.
9345 }
9346 {
9347   The~available~keys~are~(in~alphabetic~order):~
9348   footnote,~
9349   footnotehyper,~
9350   messages~for~Overleaf,~
9351   renew~dots,~and~
9352   renew~matrix.
9353 }

9354 \@@_msg_new:nn { no-test-for-array }
9355 {
9356   The~key~'no-test-for-array'~has~been~deprecated~and~will~be~
9357   deleted~in~a~future~version~of~nicematrix.
9358 }

9359 \keys_define:nn { nicematrix / Package }
9360 {
9361   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9362   renew~dots .value_forbidden:n = true ,
9363   renew~matrix .code:n = \@@_renew_matrix: ,
9364   renew~matrix .value_forbidden:n = true ,
9365   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9366   footnote .bool_set:N = \g_@@_footnote_bool ,
9367   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. We keep the following key only for compatibility but maybe we will delete it.

```

9368   no-test-for-array .code:n = \@@_warning:n { no-test-for-array } ,
9369   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9370 }
9371 \ProcessKeysOptions { nicematrix / Package }

```

```

9372 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9373 {
9374   You~can't~use~the~option~'footnote'~because~the~package~
9375   footnotehyper~has~already~been~loaded.~
9376   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~

```

```

9377     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9378     of~the~package~footnotehyper.\\
9379     The~package~footnote~won't~be~loaded.
9380   }
9381   \@@_msg_new:nn { footnotehyper~with~footnote~package }
9382   {
9383     You~can't~use~the~option~'footnotehyper'~because~the~package~
9384     footnote~has~already~been~loaded.~
9385     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9386     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9387     of~the~package~footnote.\\
9388     The~package~footnotehyper~won't~be~loaded.
9389   }
9390   \bool_if:NT \g_@@_footnote_bool
9391   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9392   \IfClassLoadedTF { beamer }
9393   { \bool_set_false:N \g_@@_footnote_bool }
9394   {
9395     \IfPackageLoadedTF { footnotehyper }
9396     { \@@_error:n { footnote~with~footnotehyper~package } }
9397     { \usepackage { footnote } }
9398   }
9399 }
9400 \bool_if:NT \g_@@_footnotehyper_bool
9401 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9402   \IfClassLoadedTF { beamer }
9403   { \bool_set_false:N \g_@@_footnote_bool }
9404   {
9405     \IfPackageLoadedTF { footnote }
9406     { \@@_error:n { footnotehyper~with~footnote~package } }
9407     { \usepackage { footnotehyper } }
9408   }
9409   \bool_set_true:N \g_@@_footnote_bool
9410 }

```

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.

```

9411 \bool_new:N \l_@@_underscore_loaded_bool
9412 \IfPackageLoadedT { underscore }
9413 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9414 \hook_gput_code:nnn { begindocument } { . }
9415 {
9416   \bool_if:NF \l_@@_underscore_loaded_bool
9417   {
9418     \IfPackageLoadedT { underscore }
9419     { \@@_error:n { underscore~after~nicematrix } }
9420   }
9421 }

```

39 Error messages of the package

```

9422 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9423 { \str_const:Nn \c_@@_available_keys_str { } }
9424 {
9425   \str_const:Nn \c_@@_available_keys_str
9426   { For~a~list~of~the~available~keys,~type~H~<return>. }
9427 }
9428 \seq_new:N \g_@@_types_of_matrix_seq
9429 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9430 {
9431   NiceMatrix ,
9432   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9433 }
9434 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9435 { \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`.

```

9436 \cs_new_protected:Npn \@@_error_too_much_cols:
9437 {
9438   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9439   { \@@_fatal:nn { too-much-cols-for-array } }
9440   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9441   { \@@_fatal:n { too-much-cols-for-matrix } }
9442   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9443   { \@@_fatal:n { too-much-cols-for-matrix } }
9444   \bool_if:NF \l_@@_last_col_without_value_bool
9445   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9446 }

```

The following command must *not* be protected since it's used in an error message.

```

9447 \cs_new:Npn \@@_message_hdotsfor:
9448 {
9449   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9450   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9451 }
9452 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9453 {
9454   Incompatible~options.\\
9455   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9456   The~output~will~not~be~reliable.
9457 }
9458 \@@_msg_new:nn { key~color~inside }
9459 {
9460   Key~deprecated.\\
9461   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9462   and~have~been~deprecated.\\
9463   You~won't~have~similar~message~till~the~end~of~the~document.
9464 }
9465 \@@_msg_new:nn { negative~weight }
9466 {
9467   Negative~weight.\\
9468   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9469   the~value~'\int_use:N \l_@@_weight_int'.\\
9470   The~absolute~value~will~be~used.
9471 }
9472 \@@_msg_new:nn { last~col~not~used }

```

```

9473 {
9474   Column~not~used.\\
9475   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9476   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9477 }
9478 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9479 {
9480   Too~much~columns.\\
9481   In~the~row~\int_eval:n { \c@iRow },~
9482   you~try~to~use~more~columns~
9483   than~allowed~by~your~\@@_full_name_env:. \@@_message_hdotsfor:\
9484   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9485   (plus~the~exterior~columns).~This~error~is~fatal.
9486 }
9487 \@@_msg_new:nn { too~much~cols~for~matrix }
9488 {
9489   Too~much~columns.\\
9490   In~the~row~\int_eval:n { \c@iRow },~
9491   you~try~to~use~more~columns~than~allowed~by~your~
9492   \@@_full_name_env:. \@@_message_hdotsfor:\ Recall~that~the~maximal~
9493   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9494   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9495   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9496   \token_to_str:N \setcounter\ to~change~that~value).~
9497   This~error~is~fatal.
9498 }
9499 \@@_msg_new:nn { too~much~cols~for~array }
9500 {
9501   Too~much~columns.\\
9502   In~the~row~\int_eval:n { \c@iRow },~
9503   ~you~try~to~use~more~columns~than~allowed~by~your~
9504   \@@_full_name_env:. \@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9505   \int_use:N \g_@@_static_num_of_col_int
9506   \bool_if:nT
9507     { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9508     { ~(plus~the~exterior~ones) }.~
9509   This~error~is~fatal.
9510 }
9511 \@@_msg_new:nn { columns~not~used }
9512 {
9513   Columns~not~used.\\
9514   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9515   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9516   The~columns~you~did~not~used~won't~be~created.\\
9517   You~won't~have~similar~error~message~till~the~end~of~the~document.
9518 }
9519 \@@_msg_new:nn { empty~preamble }
9520 {
9521   Empty~preamble.\\
9522   The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9523   This~error~is~fatal.
9524 }
9525 \@@_msg_new:nn { in~first~col }
9526 {
9527   Erroneous~use.\\
9528   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9529   That~command~will~be~ignored.
9530 }
9531 \@@_msg_new:nn { in~last~col }
9532 {

```

```

9533     Erroneous~use.\\
9534     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9535     That~command~will~be~ignored.
9536 }

9537 \\@@_msg_new:nn { in~first~row }
9538 {
9539     Erroneous~use.\\
9540     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9541     That~command~will~be~ignored.
9542 }

9543 \\@@_msg_new:nn { in~last~row }
9544 {
9545     Erroneous~use.\\
9546     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9547     That~command~will~be~ignored.
9548 }

9549 \\@@_msg_new:nn { TopRule~without~booktabs }
9550 {
9551     Erroneous~use.\\
9552     You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9553     That~command~will~be~ignored.
9554 }

9555 \\@@_msg_new:nn { TopRule~without~tikz }
9556 {
9557     Erroneous~use.\\
9558     You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9559     That~command~will~be~ignored.
9560 }

9561 \\@@_msg_new:nn { caption~outside~float }
9562 {
9563     Key~caption~forbidden.\\
9564     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9565     environment.~This~key~will~be~ignored.
9566 }

9567 \\@@_msg_new:nn { short~caption~without~caption }
9568 {
9569     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9570     However,~your~'short~caption'~will~be~used~as~'caption'.
9571 }

9572 \\@@_msg_new:nn { double~closing~delimiter }
9573 {
9574     Double~delimiter.\\
9575     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9576     delimiter.~This~delimiter~will~be~ignored.
9577 }

9578 \\@@_msg_new:nn { delimiter~after~opening }
9579 {
9580     Double~delimiter.\\
9581     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9582     delimiter.~That~delimiter~will~be~ignored.
9583 }

9584 \\@@_msg_new:nn { bad~option~for~line~style }
9585 {
9586     Bad~line~style.\\
9587     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9588     is~'standard'.~That~key~will~be~ignored.
9589 }

9590 \\@@_msg_new:nn { corners~with~no~cell~nodes }
9591 {

```



```

9592     Incompatible-keys.\\
9593     You-can't-use-the-key~'corners'~here~because~the-key~'no-cell-nodes'~
9594     is~in~force.\\
9595     If~you~go~on,~that~key~will~be~ignored.
9596 }
9597 \@@_msg_new:nn { extra-nodes~with~no~cell~nodes }
9598 {
9599     Incompatible-keys.\\
9600     You-can't~create~'extra-nodes'~here~because~the-key~'no-cell-nodes'~
9601     is~in~force.\\
9602     If~you~go~on,~those~extra~nodes~won't~be~created.
9603 }
9604 \@@_msg_new:nn { Identical-notes~in~caption }
9605 {
9606     Identical~tabular~notes.\\
9607     You-can't~put~several~notes~with~the~same~content~in~
9608     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9609     If~you~go~on,~the~output~will~probably~be~erroneous.
9610 }
9611 \@@_msg_new:nn { tabularnote~below~the~tabular }
9612 {
9613     \token_to_str:N \tabularnote\ forbidden\\
9614     You-can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9615     of~your~tabular~because~the~caption~will~be~composed~below~
9616     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9617     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9618     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9619     no~similar~error~will~raised~in~this~document.
9620 }
9621 \@@_msg_new:nn { Unknown-key~for~rules }
9622 {
9623     Unknown~key.\\
9624     There~is~only~two~keys~available~here:~width~and~color.\\
9625     Your~key~'\l_keys_key_str'~will~be~ignored.
9626 }
9627 \@@_msg_new:nn { Unknown-key~for~TikzEveryCell }
9628 {
9629     Unknown~key.\\
9630     There~is~only~two~keys~available~here:~
9631     'empty'~and~'not~empty'.\\
9632     Your~key~'\l_keys_key_str'~will~be~ignored.
9633 }
9634 \@@_msg_new:nn { Unknown-key~for~rotate }
9635 {
9636     Unknown~key.\\
9637     The~only~key~available~here~is~'c'.\\
9638     Your~key~'\l_keys_key_str'~will~be~ignored.
9639 }
9640 \@@_msg_new:nnn { Unknown-key~for~custom~line }
9641 {
9642     Unknown~key.\\
9643     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9644     It~you~go~on,~you~will~probably~have~other~errors. \\
9645     \c_@@_available_keys_str
9646 }
9647 {
9648     The~available~keys~are~(in~alphabetic~order):~
9649     ccommand,~
9650     color,~
9651     command,~
9652     dotted,~

```

```

9653     letter,~
9654     multiplicity,~
9655     sep-color,~
9656     tikz,~and~total-width.
9657 }
9658 \@@_msg_new:nnn { Unknown~key~for~xdots }
9659 {
9660     Unknown~key.\\
9661     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9662     \c_@@_available_keys_str
9663 }
9664 {
9665     The~available~keys~are~(in~alphabetic~order):~
9666     'color',~
9667     'horizontal-labels',~
9668     'inter',~
9669     'line-style',~
9670     'radius',~
9671     'shorten',~
9672     'shorten-end'~and~'shorten-start'.
9673 }
9674 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9675 {
9676     Unknown~key.\\
9677     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9678     (and~you~try~to~use~'\l_keys_key_str')\\
9679     That~key~will~be~ignored.
9680 }
9681 \@@_msg_new:nn { label~without~caption }
9682 {
9683     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9684     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9685 }
9686 \@@_msg_new:nn { W~warning }
9687 {
9688     Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9689     (row~\int_use:N \c@iRow).
9690 }
9691 \@@_msg_new:nn { Construct~too~large }
9692 {
9693     Construct~too~large.\\
9694     Your~command~\token_to_str:N #1
9695     can't~be~drawn~because~your~matrix~is~too~small.\\
9696     That~command~will~be~ignored.
9697 }
9698 \@@_msg_new:nn { underscore~after~nicematrix }
9699 {
9700     Problem~with~'underscore'.\\
9701     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9702     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9703     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{-times}}'.
9704 }
9705 \@@_msg_new:nn { ampersand~in~light~syntax }
9706 {
9707     Ampersand~forbidden.\\
9708     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9709     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9710 }
9711 \@@_msg_new:nn { double~backslash~in~light~syntax }
9712 {

```

```

9713 Double~backslash~forbidden.\\
9714 You~can't~use~\token_to_str:N
9715 \\~to~separate~rows~because~the~key~'light-syntax'~
9716 is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9717 (set~by~the~key~'end-of-row').~This~error~is~fatal.
9718 }
9719 \@@_msg_new:nn { hlines~with~color }
9720 {
9721 Incompatible~keys.\\
9722 You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9723 \token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9724 However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9725 Your~key~will~be~discarded.
9726 }
9727 \@@_msg_new:nn { bad~value~for~baseline }
9728 {
9729 Bad~value~for~baseline.\\
9730 The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9731 valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9732 \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9733 the~form~'line-i'.\\
9734 A~value~of~1~will~be~used.
9735 }
9736 \@@_msg_new:nn { detection~of~empty~cells }
9737 {
9738 Problem~with~'not~empty'\\
9739 For~technical~reasons,~you~must~activate~
9740 'create~cell~nodes'~in~\token_to_str:N \CodeBefore\
9741 in~order~to~use~the~key~'\l_keys_key_str'.\\
9742 That~key~will~be~ignored.
9743 }
9744 \@@_msg_new:nn { siunitx~not~loaded }
9745 {
9746 siunitx~not~loaded\\
9747 You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9748 That~error~is~fatal.
9749 }
9750 \@@_msg_new:nn { Invalid~name }
9751 {
9752 Invalid~name.\\
9753 You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9754 \SubMatrix\ of~your~\@@_full_name_env:.\
9755 A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9756 This~key~will~be~ignored.
9757 }
9758 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9759 {
9760 Wrong~line.\\
9761 You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9762 \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9763 number~is~not~valid.~It~will~be~ignored.
9764 }
9765 \@@_msg_new:nn { Impossible~delimiter }
9766 {
9767 Impossible~delimiter.\\
9768 It's~impossible~to~draw~the~#1~delimiter~of~your~
9769 \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9770 in~that~column.
9771 \bool_if:NT \l_@@_submatrix_slim_bool
9772 { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9773 This~\token_to_str:N \SubMatrix\ will~be~ignored.

```

```

9774 }
9775 \@@_msg_new:nnn { width-without-X-columns }
9776 {
9777   You-have-used-the-key~'width'~but~you-have-put-no~'X'~column.~
9778   That~key~will~be~ignored.
9779 }
9780 {
9781   This~message~is~the~message~'width~without~X~columns'~
9782   of~the~module~'nicematrix'.~
9783   The~experimented~users~can~disable~that~message~with~
9784   \token_to_str:N \msg_redirect_name:nnn.\\
9785 }
9786
9787 \@@_msg_new:nn { key-multiplicity-with-dotted }
9788 {
9789   Incompatible~keys. \\
9790   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9791   in~a~'custom-line'.~They~are~incompatible. \\
9792   The~key~'multiplicity'~will~be~discarded.
9793 }
9794 \@@_msg_new:nn { empty-environment }
9795 {
9796   Empty~environment.\\
9797   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9798 }
9799 \@@_msg_new:nn { No-letter-and-no-command }
9800 {
9801   Erroneous~use.\\
9802   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9803   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9804   '~'ccommand'~(to~draw~horizontal~rules).\\
9805   However,~you~can~go~on.
9806 }
9807 \@@_msg_new:nn { Forbidden-letter }
9808 {
9809   Forbidden~letter.\\
9810   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9811   It~will~be~ignored.
9812 }
9813 \@@_msg_new:nn { Several-letters }
9814 {
9815   Wrong~name.\\
9816   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9817   have~used~'\l_@@_letter_str').\\
9818   It~will~be~ignored.
9819 }
9820 \@@_msg_new:nn { Delimiter-with-small }
9821 {
9822   Delimiter~forbidden.\\
9823   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9824   because~the~key~'small'~is~in~force.\\
9825   This~error~is~fatal.
9826 }
9827 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9828 {
9829   Unknown~cell.\\
9830   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9831   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9832   can't~be~executed~because~a~cell~doesn't~exist.\\
9833   This~command~\token_to_str:N \line\ will~be~ignored.

```

```

9834 }
9835 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9836 {
9837   Duplicate-name.\
9838   The-name-#1-is-already-used-for-a-token_to_str:N \SubMatrix\
9839   in-this-@@_full_name_env:.\
9840   This-key-will-be-ignored.\
9841   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9842   { For-a-list-of-the-names-already-used,-type-H<return>. }
9843 }
9844 {
9845   The-names-already-defined-in-this-@@_full_name_env:\ are:~
9846   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9847 }
9848 \@@_msg_new:nn { r-or-l-with-preamble }
9849 {
9850   Erroneous-use.\
9851   You-can't-use-the-key-\l_keys_key_str'~in-your-@@_full_name_env:~
9852   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
9853   your-@@_full_name_env:.\
9854   This-key-will-be-ignored.
9855 }
9856 \@@_msg_new:nn { Hdotsfor-in-col-0 }
9857 {
9858   Erroneous-use.\
9859   You-can't-use-token_to_str:N \Hdotsfor\ in-an-exterior-column-of~
9860   the-array.~This-error-is-fatal.
9861 }
9862 \@@_msg_new:nn { bad-corner }
9863 {
9864   Bad-corner.\
9865   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
9866   'corners').~The-available-values-are:~NW,~SW,~NE~and~SE.\
9867   This-specification-of-corner-will-be-ignored.
9868 }
9869 \@@_msg_new:nn { bad-border }
9870 {
9871   Bad-border.\
9872   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9873   (in-the-key~'borders'~of~the-command-token_to_str:N \Block).~
9874   The-available-values-are:~left,~right,~top~and~bottom~(and-you-can~
9875   also-use-the-key~'tikz'
9876   \IfPackageLoadedF { tikz }
9877   {-if-you-load-the-LaTeX-package~'tikz'})..\
9878   This-specification-of-border-will-be-ignored.
9879 }
9880 \@@_msg_new:nn { TikzEveryCell-without-tikz }
9881 {
9882   TikZ-not-loaded.\
9883   You-can't-use-token_to_str:N \TikzEveryCell\
9884   because-you-have-not-loaded-tikz.~
9885   This-command-will-be-ignored.
9886 }
9887 \@@_msg_new:nn { tikz-key-without-tikz }
9888 {
9889   TikZ-not-loaded.\
9890   You-can't-use-the-key~'tikz'~for-the-command~'\token_to_str:N
9891   \Block'~because-you-have-not-loaded-tikz.~
9892   This-key-will-be-ignored.
9893 }

```

```

9894 \@@_msg_new:nn { last-col-non-empty-for-NiceArray }
9895 {
9896   Erroneous-use.\\
9897   In-the-\@@_full_name_env:,~you-must-use-the-key~
9898   'last-col'~without-value.\\
9899   However,~you-can-go-on-for-this-time~
9900   (the-value~'\l_keys_value_tl'~will-be-ignored).
9901 }
9902 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9903 {
9904   Erroneous-use.\\
9905   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9906   'last-col'~without-value.\\
9907   However,~you-can-go-on-for-this-time~
9908   (the-value~'\l_keys_value_tl'~will-be-ignored).
9909 }
9910 \@@_msg_new:nn { Block-too-large-1 }
9911 {
9912   Block-too-large.\\
9913   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
9914   too-small-for-that-block. \\
9915   This-block-and-maybe-others-will-be-ignored.
9916 }
9917 \@@_msg_new:nn { Block-too-large-2 }
9918 {
9919   Block-too-large.\\
9920   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9921   \g_@@_static_num_of_col_int\
9922   columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9923   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9924   (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9925   This-block-and-maybe-others-will-be-ignored.
9926 }
9927 \@@_msg_new:nn { unknown-column-type }
9928 {
9929   Bad-column-type.\\
9930   The-column-type~'#1'~in-your~\@@_full_name_env:\
9931   is-unknown. \\
9932   This-error-is-fatal.
9933 }
9934 \@@_msg_new:nn { unknown-column-type-S }
9935 {
9936   Bad-column-type.\\
9937   The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \\
9938   If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should~
9939   load-that-package. \\
9940   This-error-is-fatal.
9941 }
9942 \@@_msg_new:nn { tabularnote-forbidden }
9943 {
9944   Forbidden-command.\\
9945   You-can't-use-the-command~\token_to_str:N\tabularnote\
9946   ~here.~This-command-is-available-only-in~
9947   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9948   the-argument-of-a-command~\token_to_str:N \caption\ included~
9949   in-an-environment~{table}. \\
9950   This-command-will-be-ignored.
9951 }
9952 \@@_msg_new:nn { borders-forbidden }
9953 {
9954   Forbidden-key.\\

```

```

9955     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9956     because~the~option~'rounded-corners'~
9957     is~in~force~with~a~non-zero~value.\\
9958     This~key~will~be~ignored.
9959 }
9960 \@@_msg_new:nn { bottomrule~without~booktabs }
9961 {
9962     booktabs~not~loaded.\\
9963     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9964     loaded~'booktabs'.\\
9965     This~key~will~be~ignored.
9966 }
9967 \@@_msg_new:nn { enumitem~not~loaded }
9968 {
9969     enumitem~not~loaded.\\
9970     You~can't~use~the~command~\token_to_str:N\tabularnote\
9971     ~because~you~haven't~loaded~'enumitem'.\\
9972     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9973     ignored~in~the~document.
9974 }
9975 \@@_msg_new:nn { tikz~without~tikz }
9976 {
9977     Tikz~not~loaded.\\
9978     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9979     loaded.~If~you~go~on,~that~key~will~be~ignored.
9980 }
9981 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9982 {
9983     Tikz~not~loaded.\\
9984     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9985     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9986     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9987     use~that~custom~line.
9988 }
9989 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9990 {
9991     Tikz~not~loaded.\\
9992     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9993     command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9994     That~key~will~be~ignored.
9995 }
9996 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9997 {
9998     Erroneous~use.\\
9999     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
10000     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
10001     The~key~'color'~will~be~discarded.
10002 }
10003 \@@_msg_new:nn { Wrong~last~row }
10004 {
10005     Wrong~number.\\
10006     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
10007     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
10008     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
10009     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
10010     without~value~(more~compilations~might~be~necessary).
10011 }
10012 \@@_msg_new:nn { Yet~in~env }
10013 {
10014     Nested~environments.\\

```

```

10015     Environments-of-nicematrix-can't-be-nested.\\
10016     This-error-is-fatal.
10017   }
10018 \@@_msg_new:nn { Outside-math-mode }
10019   {
10020     Outside-math-mode.\\
10021     The-\@@_full_name_env:\ can-be-used-only-in-math-mode~
10022     (and-not-in-\token_to_str:N \vcenter).\\
10023     This-error-is-fatal.
10024   }
10025 \@@_msg_new:nn { One-letter-allowed }
10026   {
10027     Bad-name.\\
10028     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\\
10029     It-will-be-ignored.
10030   }
10031 \@@_msg_new:nn { TabularNote-in-CodeAfter }
10032   {
10033     Environment~{TabularNote}~forbidden.\\
10034     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
10035     but~*before*~the~\token_to_str:N \CodeAfter.\\
10036     This-environment~{TabularNote}~will-be-ignored.
10037   }
10038 \@@_msg_new:nn { varwidth-not-loaded }
10039   {
10040     varwidth-not-loaded.\\
10041     You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
10042     loaded.\\
10043     Your-column-will-behave-like~'p'.
10044   }
10045 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
10046   {
10047     Unknow-key.\\
10048     Your-key~'\l_keys_key_str'~is-unknown-for-a-rule.\\
10049     \c_@@_available_keys_str
10050   }
10051   {
10052     The-available-keys-are-(in-alphabetic-order):~
10053     color,~
10054     dotted,~
10055     multiplicity,~
10056     sep-color,~
10057     tikz,~and~total-width.
10058   }
10059
10060 \@@_msg_new:nnn { Unknow-key-for-Block }
10061   {
10062     Unknow-key.\\
10063     The-key~'\l_keys_key_str'~is-unknown-for-the-command~\token_to_str:N
10064     \Block.\\ It-will-be-ignored. \\
10065     \c_@@_available_keys_str
10066   }
10067   {
10068     The-available-keys-are-(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
10069     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10070     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10071     and~vlines.
10072   }
10073 \@@_msg_new:nnn { Unknow-key-for-Brace }
10074   {
10075     Unknow-key.\\

```



```

10076 The-key~'\l_keys_key_str'~is-unknown~for~the~commands~\token_to_str:N
10077 \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10078 It~will~be~ignored. \\
10079 \c_@@_available_keys_str
10080 }
10081 {
10082 The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
10083 right-shorten,~shorten~(which~fixes~both~left~shorten~and~
10084 right-shorten)~and~yshift.
10085 }
10086 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10087 {
10088 Unknown~key.\\
10089 The-key~'\l_keys_key_str'~is-unknown.\\
10090 It~will~be~ignored. \\
10091 \c_@@_available_keys_str
10092 }
10093 {
10094 The-available-keys-are~(in~alphabetic~order):~
10095 delimiters/color,~
10096 rules~(with~the~subkeys~'color'~and~'width'),~
10097 sub-matrix~(several~subkeys)~
10098 and~xdots~(several~subkeys).~
10099 The-latter~is~for~the~command~\token_to_str:N \line.
10100 }
10101 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10102 {
10103 Unknown~key.\\
10104 The-key~'\l_keys_key_str'~is-unknown.\\
10105 It~will~be~ignored. \\
10106 \c_@@_available_keys_str
10107 }
10108 {
10109 The-available-keys-are~(in~alphabetic~order):~
10110 create-cell-nodes,~
10111 delimiters/color~and~
10112 sub-matrix~(several~subkeys).
10113 }
10114 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10115 {
10116 Unknown~key.\\
10117 The-key~'\l_keys_key_str'~is-unknown.\\
10118 That-key-will-be-ignored. \\
10119 \c_@@_available_keys_str
10120 }
10121 {
10122 The-available-keys-are~(in~alphabetic~order):~
10123 'delimiters/color',~
10124 'extra-height',~
10125 'hlines',~
10126 'hvlines',~
10127 'left-xshift',~
10128 'name',~
10129 'right-xshift',~
10130 'rules'~(with~the~subkeys~'color'~and~'width'),~
10131 'slim',~
10132 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10133 and~'right-xshift').\\
10134 }
10135 \@@_msg_new:nnn { Unknown~key~for~notes }
10136 {
10137 Unknown~key.\\

```

```

10138 The-key~'\l_keys_key_str'~is-unknown.\\
10139 That-key-will-be-ignored. \\
10140 \c_@@_available_keys_str
10141 }
10142 {
10143 The-available-keys-are-(in~alphabetic~order):~
10144 bottomrule,~
10145 code-after,~
10146 code-before,~
10147 detect-duplicates,~
10148 enumitem-keys,~
10149 enumitem-keys-para,~
10150 para,~
10151 label-in-list,~
10152 label-in-tabular~and~
10153 style.
10154 }
10155 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10156 {
10157 Unknown~key.\\
10158 The-key~'\l_keys_key_str'~is-unknown~for~the~command~
10159 \token_to_str:N \RowStyle. \\
10160 That-key-will-be-ignored. \\
10161 \c_@@_available_keys_str
10162 }
10163 {
10164 The-available-keys-are-(in~alphabetic~order):~
10165 bold,~
10166 cell-space-top-limit,~
10167 cell-space-bottom-limit,~
10168 cell-space-limits,~
10169 color,~
10170 fill~(alias:~rowcolor),~
10171 nb-rows,
10172 opacity~and~
10173 rounded-corners.
10174 }
10175 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10176 {
10177 Unknown~key.\\
10178 The-key~'\l_keys_key_str'~is-unknown~for~the~command~
10179 \token_to_str:N \NiceMatrixOptions. \\
10180 That-key-will-be-ignored. \\
10181 \c_@@_available_keys_str
10182 }
10183 {
10184 The-available-keys-are-(in~alphabetic~order):~
10185 &-in-blocks,~
10186 allow-duplicate-names,~
10187 ampersand-in-blocks,~
10188 caption-above,~
10189 cell-space-bottom-limit,~
10190 cell-space-limits,~
10191 cell-space-top-limit,~
10192 code-for-first-col,~
10193 code-for-first-row,~
10194 code-for-last-col,~
10195 code-for-last-row,~
10196 corners,~
10197 custom-key,~
10198 create-extra-nodes,~
10199 create-medium-nodes,~
10200 create-large-nodes,~

```

```

10201 custom-line,~
10202 delimiters~(several~subkeys),~
10203 end-of-row,~
10204 first-col,~
10205 first-row,~
10206 hlines,~
10207 hvlines,~
10208 hvlines-except-borders,~
10209 last-col,~
10210 last-row,~
10211 left-margin,~
10212 light-syntax,~
10213 light-syntax-expanded,~
10214 matrix/columns-type,~
10215 no-cell-nodes,~
10216 notes~(several~subkeys),~
10217 nullify-dots,~
10218 pgf-node-code,~
10219 renew-dots,~
10220 renew-matrix,~
10221 respect-arraystretch,~
10222 rounded-corners,~
10223 right-margin,~
10224 rules~(with~the~subkeys~'color'~and~'width'),~
10225 small,~
10226 sub-matrix~(several~subkeys),~
10227 vlines,~
10228 xdots~(several~subkeys).
10229 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10230 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10231 {
10232   Unknown~key.\\
10233   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10234   \{NiceArray\}. \\
10235   That~key~will~be~ignored. \\
10236   \c_@@_available_keys_str
10237 }
10238 {
10239   The~available~keys~are~(in~alphabetic~order):~
10240   &~in~blocks,~
10241   ampersand~in~blocks,~
10242   b,~
10243   baseline,~
10244   c,~
10245   cell-space-bottom-limit,~
10246   cell-space-limits,~
10247   cell-space-top-limit,~
10248   code~after,~
10249   code~for~first~col,~
10250   code~for~first~row,~
10251   code~for~last~col,~
10252   code~for~last~row,~
10253   columns~width,~
10254   corners,~
10255   create~extra~nodes,~
10256   create~medium~nodes,~
10257   create~large~nodes,~
10258   extra~left~margin,~
10259   extra~right~margin,~
10260   first~col,~
10261   first~row,~

```

```

10262 hlines,~
10263 hvlines,~
10264 hvlines-except-borders,~
10265 last-col,~
10266 last-row,~
10267 left-margin,~
10268 light-syntax,~
10269 light-syntax-expanded,~
10270 name,~
10271 no-cell-nodes,~
10272 nullify-dots,~
10273 pgf-node-code,~
10274 renew-dots,~
10275 respect-arraystretch,~
10276 right-margin,~
10277 rounded-corners,~
10278 rules~(with~the~subkeys~'color'~and~'width'),~
10279 small,~
10280 t,~
10281 vlines,~
10282 xdots/color,~
10283 xdots/shorten-start,~
10284 xdots/shorten-end,~
10285 xdots/shorten-and~
10286 xdots/line-style.
10287 }

```

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`).

```

10288 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10289 {
10290   Unknown~key.\\
10291   The~key~'\l_keys_key_str'~is~unknown~for~the~
10292   \@@_full_name_env:. \\
10293   That~key~will~be~ignored. \\
10294   \c_@@_available_keys_str
10295 }
10296 {
10297   The~available~keys~are~(in~alphabetic~order):~
10298   &~in~blocks,~
10299   ampersand~in~blocks,~
10300   b,~
10301   baseline,~
10302   c,~
10303   cell-space-bottom-limit,~
10304   cell-space-limits,~
10305   cell-space-top-limit,~
10306   code-after,~
10307   code-for-first-col,~
10308   code-for-first-row,~
10309   code-for-last-col,~
10310   code-for-last-row,~
10311   columns-type,~
10312   columns-width,~
10313   corners,~
10314   create-extra-nodes,~
10315   create-medium-nodes,~
10316   create-large-nodes,~
10317   extra-left-margin,~
10318   extra-right-margin,~
10319   first-col,~
10320   first-row,~
10321   hlines,~
10322   hvlines,~

```

```

10323   hvlines-except-borders,~
10324   l,~
10325   last-col,~
10326   last-row,~
10327   left-margin,~
10328   light-syntax,~
10329   light-syntax-expanded,~
10330   name,~
10331   no-cell-nodes,~
10332   nullify-dots,~
10333   pgf-node-code,~
10334   r,~
10335   renew-dots,~
10336   respect-arraystretch,~
10337   right-margin,~
10338   rounded-corners,~
10339   rules~(with~the~subkeys~'color'~and~'width'),~
10340   small,~
10341   t,~
10342   vlines,~
10343   xdots/color,~
10344   xdots/shorten-start,~
10345   xdots/shorten-end,~
10346   xdots/shorten-and~
10347   xdots/line-style.
10348 }
10349 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10350 {
10351   Unknown~key.\\
10352   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10353   \{NiceTabular\}. \\
10354   That~key~will~be~ignored. \\
10355   \c_@@_available_keys_str
10356 }
10357 {
10358   The~available~keys~are~(in~alphabetic~order):~
10359   &~in~blocks,~
10360   ampersand~in~blocks,~
10361   b,~
10362   baseline,~
10363   c,~
10364   caption,~
10365   cell-space-bottom-limit,~
10366   cell-space-limits,~
10367   cell-space-top-limit,~
10368   code~after,~
10369   code~for~first~col,~
10370   code~for~first~row,~
10371   code~for~last~col,~
10372   code~for~last~row,~
10373   columns~width,~
10374   corners,~
10375   custom~line,~
10376   create~extra~nodes,~
10377   create~medium~nodes,~
10378   create~large~nodes,~
10379   extra~left~margin,~
10380   extra~right~margin,~
10381   first~col,~
10382   first~row,~
10383   hlines,~
10384   hvlines,~
10385   hvlines~except~borders,~

```

```

10386 label,~
10387 last-col,~
10388 last-row,~
10389 left-margin,~
10390 light-syntax,~
10391 light-syntax-expanded,~
10392 name,~
10393 no-cell-nodes,~
10394 notes~(several~subkeys),~
10395 nullify-dots,~
10396 pgf-node-code,~
10397 renew-dots,~
10398 respect-arraystretch,~
10399 right-margin,~
10400 rounded-corners,~
10401 rules~(with~the~subkeys~'color'~and~'width'),~
10402 short-caption,~
10403 t,~
10404 tabularnote,~
10405 vlines,~
10406 xdots/color,~
10407 xdots/shorten-start,~
10408 xdots/shorten-end,~
10409 xdots/shorten~and~
10410 xdots/line-style.
10411 }

10412 \@@_msg_new:nnn { Duplicate~name }
10413 {
10414 Duplicate~name.\\
10415 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10416 the~same~environment~name~twice.~You~can~go~on,~but,~
10417 maybe,~you~will~have~incorrect~results~especially~
10418 if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10419 message~again,~use~the~key~'allow-duplicate-names'~in~
10420 '\token_to_str:N \NiceMatrixOptions'.\\
10421 \bool_if:NF \g_@@_messages_for_Overleaf_bool
10422 { For~a~list~of~the~names~already~used,~type~H~<return>. }
10423 }
10424 {
10425 The~names~already~defined~in~this~document~are:~
10426 \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10427 }

10428 \@@_msg_new:nn { Option~auto~for~columns~width }
10429 {
10430 Erroneous~use.\\
10431 You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10432 That~key~will~be~ignored.
10433 }

10434 \@@_msg_new:nn { NiceTabularX~without~X }
10435 {
10436 NiceTabularX~without~X.\\
10437 You~should~not~use~{NiceTabularX}~without~X~columns.\\
10438 However,~you~can~go~on.
10439 }

10440 \@@_msg_new:nn { Preamble~forgotten }
10441 {
10442 Preamble~forgotten.\\
10443 You~have~probably~forgotten~the~preamble~of~your~
10444 \@@_full_name_env:. \\
10445 This~error~is~fatal.
10446 }

10447 \@@_msg_new:nn { Invalid~col~number }

```

```
10448 {
10449     Invalid~column~number.\\
10450     A~color~instruction~the~\token_to_str:N \CodeBefore\
10451     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10452 }
10453 \@@_msg_new:nn { Invalid~row~number }
10454 {
10455     Invalid~row~number.\\
10456     A~color~instruction~the~\token_to_str:N \CodeBefore\
10457     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10458 }
```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	19
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	74
13	The environment <code>{NiceMatrix}</code> and its variants	92
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
15	After the construction of the array	94
16	We draw the dotted lines	101
17	The actual instructions for drawing the dotted lines with Tikz	114
18	User commands available in the new environments	120
19	The command <code>\line</code> accessible in code-after	126
20	The command <code>\RowStyle</code>	128
21	Colors of cells, rows and columns	131
22	The vertical and horizontal rules	143
23	The empty corners	159
24	The environment <code>{NiceMatrixBlock}</code>	162
25	The extra nodes	163
26	The blocks	168
27	How to draw the dotted lines transparently	192
28	Automatic arrays	192
29	The redefinition of the command <code>\dotfill</code>	193
30	The command <code>\diagbox</code>	194

31	The keyword <code>\CodeAfter</code>	195
32	The delimiters in the preamble	196
33	The command <code>\SubMatrix</code>	197
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	205
35	The command <code>TikzEveryCell</code>	208
36	The command <code>\ShowCellNames</code>	210
37	We process the options at package loading	212
38	About the package underscore	213
39	Error messages of the package	214