

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

November 18, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 7.0 of `nicematrix`, at the date of 2024/11/18.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

97 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

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

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

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

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

Idem for `\CT@drs@`.

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

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

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

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

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

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

214 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

278 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
331 {
```

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

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

```
334   \fi:
```

```
335 }
```

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

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

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

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

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

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

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

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

```

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

342 \bool_new:N \l_@@_bold_row_style_bool

```

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

```

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

```

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

```

349 \tl_new:N \l_@@_code_tl

```

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

```

350 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

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

```

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

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

```

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

```

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

```

355 \bool_new:N \l_@@_in_code_after_bool

```

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

```

356 \bool_new:N \l_@@_ampersand_bool

```

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

```

357 \int_new:N \l_@@_old_iRow_int
358 \int_new:N \l_@@_old_jCol_int

```

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

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

```

359 \seq_new:N \l_@@_custom_line_commands_seq

```

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

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

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

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

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

```
362 \bool_new:N \l_@@_X_columns_aux_bool
363 \dim_new:N \l_@@_X_columns_dim
```

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

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

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

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

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

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

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

```
367 \tl_new:N \l_@@_code_before_tl
368 \bool_new:N \l_@@_code_before_bool
```

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

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

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

```
370 \dim_new:N \l_@@_x_initial_dim
371 \dim_new:N \l_@@_y_initial_dim
372 \dim_new:N \l_@@_x_final_dim
373 \dim_new:N \l_@@_y_final_dim
```

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

```
374 \dim_new:N \l_@@_tmpc_dim
375 \dim_new:N \l_@@_tmpd_dim
376 \dim_new:N \l_@@_tmpe_dim
377 \dim_new:N \l_@@_tmpf_dim

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

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

```
384 \bool_new:N \g_@@_empty_cell_bool
```

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

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

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

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

```
387 \seq_new:N \g_@@_blocks_seq
```

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

```
388 \seq_new:N \g_@@_pos_of_blocks_seq
```

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

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

```
389 \seq_new:N \g_@@_pos_of_xdots_seq
```

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

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

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

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

```
391 \clist_new:N \l_@@_corners_cells_clist
```

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

```
392 \seq_new:N \g_@@_submatrix_names_seq
```

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

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

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

```
394 \seq_new:N \g_@@_multicolumn_cells_seq
395 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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

```

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

```

The following counters will be used when drawing the rules.

```

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

```

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

```

405 \seq_new:N \g_@@_submatrix_seq

```

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

```

406 \int_new:N \g_@@_static_num_of_col_int

```

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

```

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

```

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

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

```

413 \dim_new:N \l_@@_tab_rounded_corners_dim

```

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

```

414 \tl_new:N \l_@@_color_tl

```

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

```

415 \dim_new:N \l_@@_offset_dim

```

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

```

416 \dim_new:N \l_@@_line_width_dim

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
433 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

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

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

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

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

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

Variables for the exterior rows and columns

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

- **First row**

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

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

```
438 \int_set:Nn \l_@@_first_row_int 1
```

- **First column**

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

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

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


- **Last row**

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

```
441 \int_new:N \l_@@_last_row_int
442 \int_set:Nn \l_@@_last_row_int { -2 }
```

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

```
445 \int_new:N \l_@@_last_col_int
446 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

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

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

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

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

Some utilities

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The following internal parameters are for:

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

```

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

```

5 The command `\tabularnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is before the `{tabular}`.

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

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

```
479 \newcounter { tabularnote }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

542 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
543 { \tl_if_novalue:nT { #1 } { #3 } }

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

```

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

```

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

```

587     \hbox_set:Nn \l_tmpa_box
588     {

```

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

```

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

```

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

```

595     \int_gdecr:N \c@tabularnote
596     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

```

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

```

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

```

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

```

7 The options

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

```

693 \tl_new:N \l_@@_caption_tl
694 \tl_new:N \l_@@_short_caption_tl
695 \tl_new:N \l_@@_label_tl

```

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

```

696 \bool_new:N \l_@@_caption_above_bool

```

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

```

697 \bool_new:N \l_@@_color_inside_bool

```


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

```
698 \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`).

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

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

```
701 \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.

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

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

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.

```
712 \dim_new:N \l_@@_xdots_radius_dim
713 \hook_gput_code:nnn { begindocument } { . }
714 { \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.

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

```
718 \bool_new:N \l_@@_light_syntax_bool
719 \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).

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

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

```
722 \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`).

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

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

```
724 \bool_new:N \l_@@_parallelize_diags_bool
725 \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.

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

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

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

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

```
731 \cs_new_protected:Npn \@@_reset_arraystretch:
732 { \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).

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

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

```
734 \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.

```
735 \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”.

```
736 \bool_new:N \l_@@_medium_nodes_bool
737 \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).

```
738 \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`).

```
739 \dim_new:N \l_@@_left_margin_dim
740 \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`.

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

```
743 \tl_new:N \l_@@_end_of_row_tl
744 \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 “:”.

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

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

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

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

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

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

```

775 line-style .value_required:n = true ,
776 color .tl_set:N = \l_@@_xdots_color_tl ,
777 color .value_required:n = true ,
778 radius .code:n =
779   \hook_gput_code:nnn { begindocument } { . }
780   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
781 radius .value_required:n = true ,
782 inter .code:n =
783   \hook_gput_code:nnn { begindocument } { . }
784   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
785 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}`.

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

859     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
860   }
861   { \@@_error:n { One-letter-allowed } }
862 } ,
863 vlines-in-sub-matrix .value_required:n = true ,
864 hvlines .code:n =
865   {
866     \bool_set_true:N \l_@@_hvlines_bool
867     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
868     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
869   } ,
870 hvlines-except-borders .code:n =
871   {
872     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
873     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
874     \bool_set_true:N \l_@@_hvlines_bool
875     \bool_set_true:N \l_@@_except_borders_bool
876   } ,
877 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.

```

878 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
879 renew-dots .value_forbidden:n = true ,
880 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
881 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
882 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,

```

```

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

```

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

```

902 \keys_define:nn { nicematrix / environments }
903 {
904   corners .clist_set:N = \l_@@_corners_clist ,
905   corners .default:n = { NW , SW , NE , SE } ,
906   code-before .code:n =
907     {
908       \tl_if_empty:nF { #1 }
909         {
910           \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
911           \bool_set_true:N \l_@@_code_before_bool
912         }
913     } ,
914   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}`.

```

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

```

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

```

926   \legacy_if:nF { measuring@ }
927   {
928     \str_set:Ne \l_tmpa_str { #1 }
929     \seq_if_in:NoTF \g_@@_names_seq \l_tmpa_str
930     { \@@_error:nn { Duplicate~name } { #1 } }
931     { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
932     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
933   } ,

```

```

934     name .value_required:n = true ,
935     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
936     code-after .value_required:n = true ,
937     color-inside .code:n =
938       \bool_set_true:N \l_@@_color_inside_bool
939       \bool_set_true:N \l_@@_code_before_bool ,
940     color-inside .value_forbidden:n = true ,
941     colortbl-like .meta:n = color-inside
942   }
943 \keys_define:nn { nicematrix / notes }
944 {
945   para .bool_set:N = \l_@@_notes_para_bool ,
946   para .default:n = true ,
947   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
948   code-before .value_required:n = true ,
949   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
950   code-after .value_required:n = true ,
951   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
952   bottomrule .default:n = true ,
953   style .cs_set:Np = \@@_notes_style:n #1 ,
954   style .value_required:n = true ,
955   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
956   label-in-tabular .value_required:n = true ,
957   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
958   label-in-list .value_required:n = true ,
959   enumitem-keys .code:n =
960     {
961       \hook_gput_code:nnn { begindocument } { . }
962       {
963         \IfPackageLoadedT { enumitem }
964           { \setlist* [ tabularnotes ] { #1 } }
965       }
966     } ,
967   enumitem-keys .value_required:n = true ,
968   enumitem-keys-para .code:n =
969     {
970       \hook_gput_code:nnn { begindocument } { . }
971       {
972         \IfPackageLoadedT { enumitem }
973           { \setlist* [ tabularnotes* ] { #1 } }
974       }
975     } ,
976   enumitem-keys-para .value_required:n = true ,
977   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
978   detect-duplicates .default:n = true ,
979   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
980 }
981 \keys_define:nn { nicematrix / delimiters }
982 {
983   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
984   max-width .default:n = true ,
985   color .tl_set:N = \l_@@_delimiters_color_tl ,
986   color .value_required:n = true ,
987 }

```

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

```

988 \keys_define:nn { nicematrix }
989 {
990   NiceMatrixOptions .inherit:n =
991     { nicematrix / Global } ,
992   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,

```

```

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

```

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

```

1030 \keys_define:nn { nicematrix / NiceMatrixOptions }
1031   {
1032     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1033     delimiters / color .value_required:n = true ,
1034     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1035     delimiters / max-width .default:n = true ,
1036     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1037     delimiters .value_required:n = true ,
1038     width .dim_set:N = \l_@@_width_dim ,
1039     width .value_required:n = true ,
1040     last-col .code:n =
1041       \tl_if_empty:nF { #1 }
1042         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1043       \int_zero:N \l_@@_last_col_int ,
1044     small .bool_set:N = \l_@@_small_bool ,
1045     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.

```

1046     renew-matrix .code:n = \@@_renew_matrix: ,
1047     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}`.

```
1048 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.

```
1049 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)`.

```
1050 \str_if_eq:eeTF { #1 } { auto }
1051 { \@@_error:n { Option~auto~for~columns~width } }
1052 { \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.

```
1053 allow-duplicate-names .code:n =
1054 \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1055 allow-duplicate-names .value_forbidden:n = true ,
1056 notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1057 notes .value_required:n = true ,
1058 sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1059 sub-matrix .value_required:n = true ,
1060 matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1061 matrix / columns-type .value_required:n = true ,
1062 caption-above .bool_set:N = \l_@@_caption_above_bool ,
1063 caption-above .default:n = true ,
1064 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1065 }
```

`\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.

```
1066 \NewDocumentCommand \NiceMatrixOptions { m }
1067 { \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.

```
1068 \keys_define:nn { nicematrix / NiceMatrix }
1069 {
1070 last-col .code:n = \tl_if_empty:nTF { #1 }
1071 {
1072 \bool_set_true:N \l_@@_last_col_without_value_bool
1073 \int_set:Nn \l_@@_last_col_int { -1 }
1074 }
1075 { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1076 columns-type .tl_set:N = \l_@@_columns_type_tl ,
1077 columns-type .value_required:n = true ,
1078 l .meta:n = { columns-type = l } ,
1079 r .meta:n = { columns-type = r } ,
1080 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1081 delimiters / color .value_required:n = true ,
1082 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1083 delimiters / max-width .default:n = true ,
1084 delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1085 delimiters .value_required:n = true ,
1086 small .bool_set:N = \l_@@_small_bool ,
1087 small .value_forbidden:n = true ,
1088 unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1089 }
```

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

```
1090 \keys_define:nn { nicematrix / NiceArray }
1091 {
```

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.

```
1092     small .bool_set:N = \l_@@_small_bool ,
1093     small .value_forbidden:n = true ,
1094     last-col .code:n = \tl_if_empty:nF { #1 }
1095         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1096         \int_zero:N \l_@@_last_col_int ,
1097     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1098     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1099     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1100 }
```

```
1101 \keys_define:nn { nicematrix / pNiceArray }
1102 {
1103     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1104     last-col .code:n = \tl_if_empty:nF { #1 }
1105         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1106         \int_zero:N \l_@@_last_col_int ,
1107     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1108     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1109     delimiters / color .value_required:n = true ,
1110     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1111     delimiters / max-width .default:n = true ,
1112     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1113     delimiters .value_required:n = true ,
1114     small .bool_set:N = \l_@@_small_bool ,
1115     small .value_forbidden:n = true ,
1116     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1117     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1118     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1119 }
```

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

```
1120 \keys_define:nn { nicematrix / NiceTabular }
1121 {
```

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.

```
1122     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1123         \bool_set_true:N \l_@@_width_used_bool ,
1124     width .value_required:n = true ,
1125     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1126     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1127     tabularnote .value_required:n = true ,
1128     caption .tl_set:N = \l_@@_caption_tl ,
1129     caption .value_required:n = true ,
1130     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1131     short-caption .value_required:n = true ,
1132     label .tl_set:N = \l_@@_label_tl ,
1133     label .value_required:n = true ,
1134     last-col .code:n = \tl_if_empty:nF { #1 }
1135         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1136         \int_zero:N \l_@@_last_col_int ,
1137     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1138     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1139     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1140 }
```

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

1141 \keys_define:nn { nicematrix / CodeAfter }
1142 {
1143   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1144   delimiters / color .value_required:n = true ,
1145   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1146   rules .value_required:n = true ,
1147   xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1148   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1149   sub-matrix .value_required:n = true ,
1150   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1151 }
```

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}`).

```
1152 \cs_new_protected:Npn \@@_cell_begin:
1153 {
```

`\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.

```
1154   \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).

```
1155   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1156   \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.

```
1157   \int_compare:nNnT \c@jCol = \c_one_int
1158     { \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:`.

```
1159   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1160   \@@_tuning_not_tabular_begin:
1161   \@@_tuning_first_row:
1162   \@@_tuning_last_row:
1163   \g_@@_row_style_tl
1164 }
```

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.

```

1165 \cs_new_protected:Npn \@@_tuning_first_row:
1166 {
1167   \if_int_compare:w \c@iRow = \c_zero_int
1168     \if_int_compare:w \c@jCol > \c_zero_int
1169       \l_@@_code_for_first_row_tl
1170       \xglobal \colorlet { nicematrix-first-row } { . }
1171     \fi:
1172   \fi:
1173 }

```

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.

```

1174 \cs_new_protected:Npn \@@_tuning_last_row:
1175 {
1176   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1177     \l_@@_code_for_last_row_tl
1178     \xglobal \colorlet { nicematrix-last-row } { . }
1179   \fi:
1180 }

```

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

```

1181 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1182 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1183 {
1184   \c_math_toggle_token

```

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

```

1185   \@@_tuning_key_small:
1186 }
1187 \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.

```

1188 \cs_new_protected:Npn \@@_begin_of_row:
1189 {
1190   \int_gincr:N \c@iRow
1191   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim

```

```

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

```

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.

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1286   \@@_update_max_cell_width:

```

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

```

1287   \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

1331 \cs_new_protected:Npn \@@_node_for_cell:
1332 {
1333   \pgfpicture
1334   \pgfsetbaseline \c_zero_dim
1335   \pgfrememberpicturepositiononpagetrue
1336   \pgfset { nicematrix / cell-node }
1337   \pgfnode
1338     { rectangle }
1339     { base }
1340     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

1378 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1379 {

```



```

1380 \@@_patch_node_for_cell:n
1381   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1382 }
1383 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

1407 \bool_if:nTF

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

1458   \clist_if_empty:NF \l_@@_hlines_clist
1459   {
1460     \str_if_eq:eeF \l_@@_hlines_clist { all }
1461     {
1462       \clist_if_in:NeT

```

```

1463         \l_@@_hlines_clist
1464         { \int_eval:n { \c@iRow + 1 } }
1465     }
1466     {

```

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

```

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

```

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

```

1476 \cs_set_protected:Npn \@@_renew_dots:
1477 {
1478     \cs_set_eq:NN \ldots \@@_Ldots
1479     \cs_set_eq:NN \cdots \@@_Cdots
1480     \cs_set_eq:NN \vdots \@@_Vdots
1481     \cs_set_eq:NN \ddots \@@_Ddots
1482     \cs_set_eq:NN \iddots \@@_Iddots
1483     \cs_set_eq:NN \dots \@@_Ldots
1484     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1485 }
1486 \cs_new_protected:Npn \@@_test_color_inside:
1487 {
1488     \bool_if:NF \l_@@_color_inside_bool
1489     {

```

We will issue an error only during the first run.

```

1490         \bool_if:NF \g_@@_aux_found_bool
1491         { \@@_error:n { without~color~inside } }
1492     }
1493 }

```

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

```

1494 \hook_gput_code:nnn { begindocument } { . }
1495 {
1496     \IfPackageLoadedTF { colortbl }
1497     {
1498         \cs_set_protected:Npn \@@_everycr:
1499         { \CT@everycr { \noalign { \@@_in_everycr: } } }
1500     }
1501     {
1502         \cs_new_protected:Npn \@@_everycr:
1503         { \everycr { \noalign { \@@_in_everycr: } } }
1504     }
1505 }

```

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.

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

```

1506 \hook_gput_code:nnn { begindocument } { . }
1507 {
1508   \IfPackageLoadedTF { booktabs }
1509   {
1510     \cs_new_protected:Npn \@@_patch_booktabs:
1511       { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1512   }
1513   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1514 }

```

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`.

```

1515 \cs_new_protected:Npn \@@_some_initialization:
1516 {
1517   \@@_everycr:
1518   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1519   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1520   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1521   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1522   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1523   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1524 }

```

```

1525 \cs_new_protected:Npn \@@_pre_array_ii:
1526 {

```

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

```

1527   \int_gzero:N \g_@@_total_X_weight_int
1528   \@@_expand_clist:N \l_@@_hlines_clist
1529   \@@_expand_clist:N \l_@@_vlines_clist
1530   \@@_patch_booktabs:
1531   \box_clear_new:N \l_@@_cell_box
1532   \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}`).

```

1533   \bool_if:NT \l_@@_small_bool
1534   {
1535     \cs_set_nopar:Npn \arraystretch { 0.47 }
1536     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1537     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1538   }

```

```

1539   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1540   {
1541     \tl_put_right:Nn \@@_begin_of_row:
1542     {
1543       \pgfsys@markposition

```

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

```

1544         { \@@_env: - row - \int_use:N \c@iRow - base }
1545     }
1546 }

```

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`.

```

1547 \bool_if:nTF
1548 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1549 {
1550     \cs_set_nopar:Npn \ar@ialign
1551     {
1552         \bool_if:NT \c_@@_testphase_table_bool
1553         \tbl_init_cell_data_for_table:
1554         \@@_some_initialization:
1555         \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`.

```

1556         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1557         \halign
1558     }
1559 }

```

The following part will 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`.

```

1560 {
1561     \cs_set_nopar:Npn \ialign
1562     {
1563         \@@_some_initialization:
1564         \dim_zero:N \tabskip
1565         \cs_set_eq:NN \ialign \@@_old_ialign:
1566         \halign
1567     }
1568 }

```

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).

```

1569 \cs_set_eq:NN \@@_old_ldots \ldots
1570 \cs_set_eq:NN \@@_old_cdots \cdots
1571 \cs_set_eq:NN \@@_old_vdots \vdots
1572 \cs_set_eq:NN \@@_old_ddots \ddots
1573 \cs_set_eq:NN \@@_old_iddots \iddots
1574 \bool_if:NTF \l_@@_standard_cline_bool
1575 { \cs_set_eq:NN \cline \@@_standard_cline }
1576 { \cs_set_eq:NN \cline \@@_cline }
1577 \cs_set_eq:NN \Ldots \@@_Ldots
1578 \cs_set_eq:NN \Cdots \@@_Cdots
1579 \cs_set_eq:NN \Vdots \@@_Vdots
1580 \cs_set_eq:NN \Ddots \@@_Ddots
1581 \cs_set_eq:NN \Iddots \@@_Iddots
1582 \cs_set_eq:NN \Hline \@@_Hline:
1583 \cs_set_eq:NN \Hspace \@@_Hspace:
1584 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1585 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1586 \cs_set_eq:NN \Block \@@_Block:
1587 \cs_set_eq:NN \rotate \@@_rotate:
1588 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1589 \cs_set_eq:NN \dotfill \@@_dotfill:
1590 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:

```

```

1591 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1592 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1593 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1594 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1595   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1596 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1597 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1598 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1599 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1600 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1601   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1602 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1603   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1604 \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:`.

```

1605 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1606 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1607   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1608 \@@_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!).

```

1609 \tl_if_exist:NT \l_@@_note_in_caption_tl
1610   {
1611     \tl_if_empty:NF \l_@@_note_in_caption_tl
1612     {
1613       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1614       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1615     }
1616   }

```

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).

```

1617 \seq_gclear:N \g_@@_multicolumn_cells_seq
1618 \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).

```

1619 \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`).

```

1620 \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.

```

1621 \int_gzero_new:N \g_@@_col_total_int
1622 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1623 \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.

```

1624 \tl_gclear_new:N \g_@@_Cdots_lines_tl

```

```

1625 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1626 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1627 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1628 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1629 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1630 \tl_gclear:N \g_nicematrix_code_before_tl
1631 \tl_gclear:N \g_@@_pre_code_before_tl
1632 }

```

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

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

```

1633 \cs_new_protected:Npn \@@_pre_array:
1634 {
1635   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1636   \int_gzero_new:N \c@iRow
1637   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1638   \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).

```

1639 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1640 {
1641   \bool_set_true:N \l_@@_last_row_without_value_bool
1642   \bool_if:NT \g_@@_aux_found_bool
1643     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1644 }
1645 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1646 {
1647   \bool_if:NT \g_@@_aux_found_bool
1648     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1649 }

```

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”.

```

1650 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1651 {
1652   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1653     {
1654       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1655         { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1656       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1657         { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1658     }
1659 }

1660 \seq_gclear:N \g_@@_cols_vlism_seq
1661 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1662 \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.

```

1663 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```
1664 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1665 \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.

```
1666 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1667 \@@_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.

```
1668 \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]`).

```
1669 \dim_zero_new:N \l_@@_left_delim_dim
1670 \dim_zero_new:N \l_@@_right_delim_dim
1671 \bool_if:NTF \g_@@_delims_bool
1672 {
```

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

```
1673 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1674 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1675 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1676 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1677 }
1678 {
1679 \dim_gset:Nn \l_@@_left_delim_dim
1680 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1681 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1682 }
```

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).

```
1683 \hbox_set:Nw \l_@@_the_array_box
1684 \bool_if:NT \c_@@_recent_array_bool
1685 { \UseTaggingSocket { tbl / hmode / begin } }
1686 \skip_horizontal:N \l_@@_left_margin_dim
1687 \skip_horizontal:N \l_@@_extra_left_margin_dim
```

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).

```
1688 \m@th
1689 \c_math_toggle_token
1690 \bool_if:NTF \l_@@_light_syntax_bool
1691 { \use:c { @@-light-syntax } }
1692 { \use:c { @@-normal-syntax } }
1693 }
```


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

```

1694 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1695 {
1696   \tl_set:Nn \l_tmpa_tl { #1 }
1697   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1698     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1699   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1700   \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...

```

1701   \@@_pre_array:
1702 }

```

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).

```

1703 \cs_new_protected:Npn \@@_pre_code_before:
1704 {

```

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).

```

1705   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1706   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1707   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1708   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

1709   \pgfsys@markposition { \@@_env: - position }
1710   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1711   \pgfpicture
1712   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1713   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1714   {
1715     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1716     \pgfcoordinate { \@@_env: - row - ##1 }
1717     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1718   }

```

Now, the recreation of the `col` nodes.

```

1719   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1720   {
1721     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1722     \pgfcoordinate { \@@_env: - col - ##1 }
1723     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1724   }

```

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

```

1725   \@@_create_diag_nodes:

```

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

```
1726 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1727 \endpgfpicture
```

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

```
1728 \@@_create_blocks_nodes:
1729 \IfPackageLoadedT { tikz }
1730 {
1731   \tikzset
1732   {
1733     every-picture / .style =
1734     { overlay , name-prefix = \@@_env: - }
1735   }
1736 }
1737 \cs_set_eq:NN \cellcolor \@@_cellcolor
1738 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1739 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1740 \cs_set_eq:NN \rowcolor \@@_rowcolor
1741 \cs_set_eq:NN \rowcolors \@@_rowcolors
1742 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1743 \cs_set_eq:NN \arraycolor \@@_arraycolor
1744 \cs_set_eq:NN \columncolor \@@_columncolor
1745 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1746 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1747 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1748 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1749 }
```

```
1750 \cs_new_protected:Npn \@@_exec_code_before:
1751 {
```

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...

```
1752 \clist_map_inline:Nn \l_@@_corners_cells_clist
1753 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1754 \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`.

```
1755 \@@_add_to_colors_seq:n { { nocolor } } { }
1756 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1757 \group_begin:
```

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

```
1758 \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`).

```
1759 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1760 { \@@_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`.

```
1761 \exp_last_unbraced:No \@@_CodeBefore_keys:
1762 \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.

```

1763     \@@_actually_color:
1764     \l_@@_code_before_tl
1765     \q_stop
1766     \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1767     \group_end:
1768     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1769     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1770 }

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

1781 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1782 {
1783   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1784   \@@_CodeBefore:w
1785 }

```

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.

```

1786 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1787 {
1788   \bool_if:NT \g_@@_aux_found_bool
1789   {
1790     \@@_pre_code_before:
1791     #1
1792   }
1793 }

```

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.

```

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

```

```

1811         \@@_node_position_i:
1812         \@@_pgf_rect_node:nnn
1813         { \@@_env: - ##1 - #####1 }
1814         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1815         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1816     }
1817 }
1818 }
1819 \int_step_inline:nn \c@iRow
1820 {
1821     \pgfnodealias
1822     { \@@_env: - ##1 - last }
1823     { \@@_env: - ##1 - \int_use:N \c@jCol }
1824 }
1825 \int_step_inline:nn \c@jCol
1826 {
1827     \pgfnodealias
1828     { \@@_env: - last - ##1 }
1829     { \@@_env: - \int_use:N \c@iRow - ##1 }
1830 }
1831 \@@_create_extra_nodes:
1832 }

```

```

1833 \cs_new_protected:Npn \@@_create_blocks_nodes:
1834 {
1835     \pgfpicture
1836     \pgf@relevantforpicturesizefalse
1837     \pgfrememberpicturepositiononpagetrue
1838     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1839     { \@@_create_one_block_node:nnnnn ##1 }
1840     \endpgfpicture
1841 }

```

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

```

1842 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1843 {
1844     \tl_if_empty:nF { #5 }
1845     {
1846         \@@_qpoint:n { col - #2 }
1847         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1848         \@@_qpoint:n { #1 }
1849         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1850         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1851         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1852         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1853         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1854         \@@_pgf_rect_node:nnnnn
1855         { \@@_env: - #5 }
1856         { \dim_use:N \l_tmpa_dim }
1857         { \dim_use:N \l_tmpb_dim }
1858         { \dim_use:N \l_@@_tmpc_dim }
1859         { \dim_use:N \l_@@_tmpd_dim }
1860     }
1861 }

```

```

1862 \cs_new_protected:Npn \@@_patch_for_revtext:
1863 {

```

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

```

1864 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1865 \cs_set_eq:NN \@array \@array@array
1866 \cs_set_eq:NN \@tabular \@tabular@array
1867 \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1868 \cs_set_eq:NN \array \array@array
1869 \cs_set_eq:NN \endarray \endarray@array
1870 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1871 \cs_set_eq:NN \@mkpream \@mkpream@array
1872 \cs_set_eq:NN \@classx \@classx@array
1873 \cs_set_eq:NN \insert@column \insert@column@array
1874 \cs_set_eq:NN \@arraycr \@arraycr@array
1875 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1876 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1877 }

```

10 The environment `{NiceArrayWithDelims}`

```

1878 \NewDocumentEnvironment { NiceArrayWithDelims }
1879 { m m 0 { } m ! 0 { } t \CodeBefore }
1880 {
1881 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1882 \@@_provide_pgfsyspdfmark:
1883 \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.

```

1884 \bgroup

1885 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1886 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1887 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1888 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1889 \int_gzero:N \g_@@_block_box_int
1890 \dim_zero:N \g_@@_width_last_col_dim
1891 \dim_zero:N \g_@@_width_first_col_dim
1892 \bool_gset_false:N \g_@@_row_of_col_done_bool
1893 \str_if_empty:NT \g_@@_name_env_str
1894 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1895 \bool_if:NTF \l_@@_tabular_bool
1896 \mode_leave_vertical:
1897 \@@_test_if_math_mode:
1898 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1899 \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.

```

1900 \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.

```

1901 \cs_if_exist:NT \tikz@library@external@loaded

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1902     {
1903         \tikzexternaldisable
1904         \cs_if_exist:NT \ifstandalone
1905         { \tikzset { external / optimize = false } }
1906     }

```

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

```

1907     \int_gincr:N \g_@@_env_int
1908     \bool_if:NF \l_@@_block_auto_columns_width_bool
1909     { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1910     \seq_gclear:N \g_@@_blocks_seq
1911     \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`.

```

1912     \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1913     \seq_gclear:N \g_@@_pos_of_xdots_seq
1914     \tl_gclear_new:N \g_@@_code_before_tl
1915     \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.

```

1916     \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1917     {
1918         \bool_gset_true:N \g_@@_aux_found_bool
1919         \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1920     }
1921     { \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.

```

1922     \tl_gclear:N \g_@@_aux_tl
1923     \tl_if_empty:NF \g_@@_code_before_tl
1924     {
1925         \bool_set_true:N \l_@@_code_before_bool
1926         \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1927     }
1928     \tl_if_empty:NF \g_@@_pre_code_before_tl
1929     { \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`.

```

1930     \bool_if:NTF \g_@@_delims_bool
1931     { \keys_set:nn { nicematrix / pNiceArray } }
1932     { \keys_set:nn { nicematrix / NiceArray } }
1933     { #3 , #5 }

1934     \@@_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:.`

```

1935     \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1936 }

```

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

```

1937 {

```

```

1938 \bool_if:NTF \l_@@_light_syntax_bool
1939   { \use:c { end @@-light-syntax } }
1940   { \use:c { end @@-normal-syntax } }
1941 \c_math_toggle_token
1942 \skip_horizontal:N \l_@@_right_margin_dim
1943 \skip_horizontal:N \l_@@_extra_right_margin_dim
1944
1945 % awful workaround
1946 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1947   {
1948     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1949     {
1950       \skip_horizontal:N - \l_@@_columns_width_dim
1951       \bool_if:NTF \l_@@_tabular_bool
1952         { \skip_horizontal:n { - 2 \tabcolsep } }
1953         { \skip_horizontal:n { - 2 \arraycolsep } }
1954     }
1955   }
1956 \hbox_set_end:

```

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

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

```

1957 \bool_if:NT \l_@@_width_used_bool
1958   {
1959     \int_if_zero:nT \g_@@_total_X_weight_int
1960     { \@@_error_or_warning:n { width~without~X~columns } }
1961   }

```

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 .

```

1962 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1963   {
1964     \tl_gput_right:Ne \g_@@_aux_tl
1965     {
1966       \bool_set_true:N \l_@@_X_columns_aux_bool
1967       \dim_set:Nn \l_@@_X_columns_dim
1968       {
1969         \dim_compare:nNnTF
1970           {
1971             \dim_abs:n
1972             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1973           }
1974           <
1975           { 0.001 pt }
1976           { \dim_use:N \l_@@_X_columns_dim }
1977           {
1978             \dim_eval:n
1979             {
1980               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1981               / \int_use:N \g_@@_total_X_weight_int
1982               + \l_@@_X_columns_dim
1983             }
1984           }
1985       }
1986     }
1987   }

```

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).

```

1988 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1989   {

```

```

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

```

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

```

1999     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2000     \bool_if:NTF \g_@@_last_col_found_bool
2001     { \int_gdecr:N \c@jCol }
2002     {
2003         \int_compare:nNnT \l_@@_last_col_int > { -1 }
2004         { \@@_error:n { last-col-not-used } }
2005     }

```

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

```

2006     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2007     \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2008     \int_if_zero:nT \l_@@_first_col_int
2009     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2010     \bool_if:nTF { ! \g_@@_delims_bool }
2011     {
2012         \str_if_eq:eeTF \l_@@_baseline_tl { c }
2013         \@@_use_arraybox_with_notes_c:
2014         {
2015             \str_if_eq:eeTF \l_@@_baseline_tl { b }
2016             \@@_use_arraybox_with_notes_b:
2017             \@@_use_arraybox_with_notes:
2018         }
2019     }

```

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).

```

2020     {
2021         \int_if_zero:nTF \l_@@_first_row_int
2022         {
2023             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2024             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2025         }
2026         { \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

```

2027     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2028     {
2029         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2030         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2031     }
2032     { \dim_zero:N \l_tmpb_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).


```

2033     \hbox_set:Nn \l_tmpa_box
2034     {
2035         \c_math_toggle_token
2036         \@@_color:o \l_@@_delimiters_color_tl
2037         \exp_after:wN \left \g_@@_left_delim_tl
2038         \center
2039         {

```

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.

```

2040         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2041         \hbox
2042         {
2043             \bool_if:NTF \l_@@_tabular_bool
2044             { \skip_horizontal:N -\tabcolsep }
2045             { \skip_horizontal:N -\arraycolsep }
2046             \@@_use_arraybox_with_notes_c:
2047             \bool_if:NTF \l_@@_tabular_bool
2048             { \skip_horizontal:N -\tabcolsep }
2049             { \skip_horizontal:N -\arraycolsep }
2050         }

```

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

```

2051         \skip_vertical:N -\l_tmpb_dim
2052         \skip_vertical:N \arrayrulewidth
2053     }
2054     \exp_after:wN \right \g_@@_right_delim_tl
2055     \c_math_toggle_token
2056 }

```

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.

```

2057     \bool_if:NTF \l_@@_delimiters_max_width_bool
2058     {
2059         \@@_put_box_in_flow_bis:nn
2060         \g_@@_left_delim_tl
2061         \g_@@_right_delim_tl
2062     }
2063     \@@_put_box_in_flow:
2064 }

```

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

```

2065     \bool_if:NT \g_@@_last_col_found_bool
2066     { \skip_horizontal:N \g_@@_width_last_col_dim }
2067     \bool_if:NT \l_@@_preamble_bool
2068     {
2069         \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2070         { \@@_warning_gredirect_none:n { columns-not-used } }
2071     }
2072     \@@_after_array:

```

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

```

2073     \egroup

```

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

```

2074     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2075     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2076     \iow_now:Ne \@mainaux
2077     {
2078         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2079         { \exp_not:o \g_@@_aux_tl }
2080     }

```

```

2081 \iow_now:Nn \@mainaux { \ExplSyntaxOff }
2082 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2083 }

```

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

11 Construction of the preamble of the array

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

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

```

2084 \cs_new_protected:Npn \@_transform_preamble:
2085 {
2086   \@_transform_preamble_i:
2087   \@_transform_preamble_ii:
2088 }
2089 \cs_new_protected:Npn \@_transform_preamble_i:
2090 {
2091   \int_gzero:N \c@jCol

```

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

```

2092 \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.

```

2093 \bool_gset_false:N \g_tmpb_bool

```

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

```

2094 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2095 \int_zero:N \l_tmpa_int
2096 \tl_gclear:N \g_@@_array_preamble_tl
2097 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2098 {
2099   \tl_gset:Nn \g_@@_array_preamble_tl
2100   { ! { \skip_horizontal:N \arrayrulewidth } }
2101 }
2102 {
2103   \clist_if_in:NnT \l_@@_vlines_clist 1
2104   {
2105     \tl_gset:Nn \g_@@_array_preamble_tl
2106     { ! { \skip_horizontal:N \arrayrulewidth } }
2107   }
2108 }

```

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

```

2109 \exp_last_unbraced:No \@_rec_preamble:n \g_@@_user_preamble_tl \@_stop:
2110 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2111 \@_replace_columncolor:
2112 }

```

```

2113 \hook_gput_code:nnn { begindocument } { . }
2114 {
2115   \IfPackageLoadedTF { colortbl }
2116   {

```

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).

```

2117     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2118     \cs_new_protected:Npn \@@_replace_columncolor:
2119     {
2120       \regex_replace_all:NnN
2121       \c_@@_columncolor_regex
2122       { \c { @@_columncolor_preamble } }
2123       \g_@@_array_preamble_tl
2124     }
2125   }
2126   {
2127     \cs_new_protected:Npn \@@_replace_columncolor:
2128     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2129   }
2130 }

```

```

2131 \cs_new_protected:Npn \@@_transform_preamble_ii:
2132 {

```

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

```

2133   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2134   {
2135     \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2136     { \bool_gset_true:N \g_@@_delims_bool }
2137   }
2138   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2139   \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).

```

2140   \int_if_zero:nTF \l_@@_first_col_int
2141   { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2142   {
2143     \bool_if:NF \g_@@_delims_bool
2144     {
2145       \bool_if:NF \l_@@_tabular_bool
2146       {
2147         \clist_if_empty:NT \l_@@_vlines_clist
2148         {
2149           \bool_if:NF \l_@@_exterior_arraycolsep_bool
2150           { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2151         }
2152       }
2153     }
2154   }
2155   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2156   { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2157   {
2158     \bool_if:NF \g_@@_delims_bool
2159     {
2160       \bool_if:NF \l_@@_tabular_bool
2161       {
2162         \clist_if_empty:NT \l_@@_vlines_clist

```

```

2163         {
2164             \bool_if:NF \l_@@_exterior_arraycolsep_bool
2165             { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2166         }
2167     }
2168 }
2169 }

```

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`).

```

2170 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2171 {

```

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.

```

2172     \bool_if:NF \c_@@_testphase_table_bool
2173     {
2174         \tl_gput_right:Nn \g_@@_array_preamble_tl
2175         { > { \@@_error_too_much_cols: } l }
2176     }
2177 }
2178 }

```

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.

```

2179 \cs_new_protected:Npn \@@_rec_preamble:n #1
2180 {

```

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

```

2181     \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2182     { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2183     {

```

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

```

2184         \cs_if_exist:cTF { NC @ find @ #1 }
2185         {
2186             \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2187             \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2188         }
2189         {
2190             \str_if_eq:nnTF { #1 } { S }
2191             { \@@_fatal:n { unknown~column~type~S } }
2192             { \@@_fatal:nn { unknown~column~type } { #1 } }
2193         }
2194     }
2195 }

```

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

```

2196 \cs_new_protected:Npn \@@_c #1
2197 {
2198     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2199     \tl_gclear:N \g_@@_pre_cell_tl
2200     \tl_gput_right:Nn \g_@@_array_preamble_tl
2201     { > \@@_cell_begin: c < \@@_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`.

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

```

2202   \int_gincr:N \c@jCol
2203   \@@_rec_preamble_after_col:n
2204 }

2205 \cs_new_protected:Npn \@@_l #1
2206 {
2207   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2208   \tl_gclear:N \g_@@_pre_cell_tl
2209   \tl_gput_right:Nn \g_@@_array_preamble_tl
2210   {
2211     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2212     l
2213     < \@@_cell_end:
2214   }
2215   \int_gincr:N \c@jCol
2216   \@@_rec_preamble_after_col:n
2217 }

2218 \cs_new_protected:Npn \@@_r #1
2219 {
2220   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2221   \tl_gclear:N \g_@@_pre_cell_tl
2222   \tl_gput_right:Nn \g_@@_array_preamble_tl
2223   {
2224     > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2225     r
2226     < \@@_cell_end:
2227   }
2228   \int_gincr:N \c@jCol
2229   \@@_rec_preamble_after_col:n
2230 }

```

For ! and @

```

2231 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2232 {
2233   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2234   \@@_rec_preamble:n
2235 }
2236 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2237 \cs_new_protected:cpn { @@ _ | } #1
2238 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2239   \int_incr:N \l_tmpa_int
2240   \@@_make_preamble_i_i:n
2241 }

2242 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2243 {
2244   \str_if_eq:nnTF { #1 } { | }
2245   { \use:c { @@ _ | } | }
2246   { \@@_make_preamble_i_ii:n { } #1 }
2247 }

2248 \cs_new_protected:Npn \@@_make_preamble_i_ii:n #1 #2
2249 {
2250   \str_if_eq:nnTF { #2 } { [ ]
2251   { \@@_make_preamble_i_ii:nw { #1 } [ ]
2252   { \@@_make_preamble_i_iii:n { #2 } { #1 } }
2253   }
2254 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2255 { \@@_make_preamble_i_ii:n { #1 , #2 } }

```

```

2256 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2257 {
2258   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2259   \tl_gput_right:Ne \g_@@_array_preamble_tl
2260   {

```

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

```

2261     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2262   }
2263   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2264   {
2265     \@@_vline:n
2266     {
2267       position = \int_eval:n { \c@jCol + 1 } ,
2268       multiplicity = \int_use:N \l_tmpa_int ,
2269       total-width = \dim_use:N \l_@@_rule_width_dim ,
2270       #2
2271     }

```

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.

```

2272   }
2273   \int_zero:N \l_tmpa_int
2274   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2275   \@@_rec_preamble:n #1
2276 }

```

```

2277 \cs_new_protected:cpn { @@_ > } #1 #2
2278 {
2279   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2280   \@@_rec_preamble:n
2281 }
2282 \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.

```

2283 \keys_define:nn { nicematrix / p-column }
2284 {
2285   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2286   r .value_forbidden:n = true ,
2287   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2288   c .value_forbidden:n = true ,
2289   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2290   l .value_forbidden:n = true ,
2291   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2292   S .value_forbidden:n = true ,
2293   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2294   p .value_forbidden:n = true ,
2295   t .meta:n = p ,
2296   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2297   m .value_forbidden:n = true ,
2298   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2299   b .value_forbidden:n = true
2300 }

```

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

```

2301 \cs_new_protected:Npn \@@_p #1
2302 {
2303   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2304 \@@_make_preamble_ii_i:n
2305 }
2306 \cs_set_eq:NN \@@_b \@@_p
2307 \cs_set_eq:NN \@@_m \@@_p
2308 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2309 {
2310 \str_if_eq:nnTF { #1 } { [ ]
2311 { \@@_make_preamble_ii_ii:w [ ]
2312 { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2313 }
2314 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2315 { \@@_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.

```

2316 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2317 {

```

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).

```

2318 \str_set:Nn \l_@@_hpos_col_str { j }
2319 \@@_keys_p_column:n { #1 }
2320 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2321 }
2322 \cs_new_protected:Npn \@@_keys_p_column:n #1
2323 { \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.

```

2324 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2325 {
2326 \use:e
2327 {
2328 \@@_make_preamble_ii_v:nnnnnnnn
2329 { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2330 { \dim_eval:n { #1 } }
2331 {

```

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.

```

2332 \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2333 { \t1_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2334 {

```

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

```

2335 \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2336 { \str_lowercase:o \l_@@_hpos_col_str }
2337 }
2338 \IfPackageLoadedTF { ragged2e }
2339 {
2340 \str_case:on \l_@@_hpos_col_str
2341 {
2342 c { \exp_not:N \Centering }
2343 l { \exp_not:N \RaggedRight }
2344 r { \exp_not:N \RaggedLeft }
2345 }
2346 }
2347 {
2348 \str_case:on \l_@@_hpos_col_str

```

```

2349     {
2350         c { \exp_not:N \centering }
2351         l { \exp_not:N \raggedright }
2352         r { \exp_not:N \raggedleft }
2353     }
2354 }
2355 #3
2356 }
2357 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2358 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2359 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2360 { #2 }
2361 {
2362     \str_case:onF \l_@@_hpos_col_str
2363     {
2364         { j } { c }
2365         { si } { c }
2366     }

```

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

```

2367     { \str_lowercase:o \l_@@_hpos_col_str }
2368 }
2369 }

```

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

```

2370     \int_gincr:N \c@jCol
2371     \@@_rec_preamble_after_col:n
2372 }

```

#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*.

```

2373 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2374 {
2375     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2376     {
2377         \tl_gput_right:Nn \g_@@_array_preamble_tl
2378         { > \@@_test_if_empty_for_S: }
2379     }
2380     {
2381         \tl_gput_right:Nn \g_@@_array_preamble_tl
2382         { > \@@_test_if_empty: }
2383     }
2384     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2385     \tl_gclear:N \g_@@_pre_cell_tl
2386     \tl_gput_right:Nn \g_@@_array_preamble_tl
2387     {
2388         > {

```

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.

```

2389         \dim_set:Nn \l_@@_col_width_dim { #2 }
2390         \bool_if:NT \c_@@_testphase_table_bool

```



```

2391         { \tag_struct_begin:n { tag = Div } }
2392     \@@_cell_begin:

```

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

```

2393     \use:c { #7 } [ #1 ] { #2 }

```

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

```

2394     \everypar
2395     {
2396         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2397         \everypar { }
2398     }
2399     \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.).

```

2400     #3

```

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

```

2401     \g_@@_row_style_tl
2402     \arraybackslash
2403     #5
2404     }
2405     #8
2406     < {
2407     #6

```

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

```

2408     \@finalstrut \@arstrutbox
2409     \use:c { end #7 }

```

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

```

2410     #4
2411     \@@_cell_end:
2412     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2413     }
2414     }
2415     }

```

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

```

2416 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2417 {

```

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 `&`).

```

2418 \group_align_safe_begin:
2419 \peek_meaning:NTF &
2420 {
2421     \group_align_safe_end:
2422     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2423     {

```

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

```

2424     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2425     \skip_horizontal:N \l_@@_col_width_dim
2426     }
2427     }
2428     { \group_align_safe_end: }
2429     }

```

```

2430 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2431 {
2432   \peek_meaning:NT \_siunitx_table_skip:n
2433   { \bool_gset_true:N \g_@@_empty_cell_bool }
2434 }

```

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.

```

2435 \cs_new_protected:Npn \@@_center_cell_box:
2436 {

```

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

```

2437   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2438   {
2439     \int_compare:nNnT
2440     { \box_ht:N \l_@@_cell_box }
2441     >

```

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).

```

2442     { \box_ht:N \strutbox }
2443     {
2444       \hbox_set:Nn \l_@@_cell_box
2445       {
2446         \box_move_down:nn
2447         {
2448           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2449             + \baselineskip ) / 2
2450         }
2451         { \box_use:N \l_@@_cell_box }
2452       }
2453     }
2454 }
2455 }

```

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

```

2456 \cs_new_protected:Npn \@@_V #1 #2
2457 {
2458   \str_if_eq:nnTF { #1 } { [ ]
2459     { \@@_make_preamble_V_i:w [ ]
2460       { \@@_make_preamble_V_i:w [ ] { #2 } }
2461     }
2462   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2463   { \@@_make_preamble_V_ii:nn { #1 } }
2464   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2465   {
2466     \str_set:Nn \l_@@_vpos_col_str { p }
2467     \str_set:Nn \l_@@_hpos_col_str { j }
2468     \@@_keys_p_column:n { #1 }
2469     \IfPackageLoadedTF { varwidth }
2470     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2471     {
2472       \@@_error_or_warning:n { varwidth-not-loaded }
2473       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2474     }
2475   }

```

For w and W

```
2476 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2477 \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.

```
2478 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2479 {
2480   \str_if_eq:nnTF { #3 } { s }
2481   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2482   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2483 }
```

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.

```
2484 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2485 {
2486   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2487   \tl_gclear:N \g_@@_pre_cell_tl
2488   \tl_gput_right:Nn \g_@@_array_preamble_tl
2489   {
2490     > {
2491       \dim_set:Nn \l_@@_col_width_dim { #2 }
2492       \@@_cell_begin:
2493       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2494     }
2495     c
2496     < {
2497       \@@_cell_end_for_w_s:
2498       #1
2499       \@@_adjust_size_box:
2500       \box_use_drop:N \l_@@_cell_box
2501     }
2502   }
2503   \int_gincr:N \c@jCol
2504   \@@_rec_preamble_after_col:n
2505 }
```

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

```
2506 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2507 {
2508   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2509   \tl_gclear:N \g_@@_pre_cell_tl
2510   \tl_gput_right:Nn \g_@@_array_preamble_tl
2511   {
2512     > {
```

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.

```
2513       \dim_set:Nn \l_@@_col_width_dim { #4 }
2514       \hbox_set:Nw \l_@@_cell_box
2515       \@@_cell_begin:
2516       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2517     }
2518     c
2519     < {
2520       \@@_cell_end:
2521       \hbox_set_end:
2522       #1
```

```

2523         \@@_adjust_size_box:
2524         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2525     }
2526 }

```

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

```

2527     \int_gincr:N \c@jCol
2528     \@@_rec_preamble_after_col:n
2529 }

```

```

2530 \cs_new_protected:Npn \@@_special_W:
2531 {
2532     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2533     { \@@_warning:n { W~warning } }
2534 }

```

For S (of siunitx).

```

2535 \cs_new_protected:Npn \@@_S #1 #2
2536 {
2537     \str_if_eq:nnTF { #2 } { [ ]
2538         { \@@_make_preamble_S:w [ ] }
2539         { \@@_make_preamble_S:w [ ] { #2 } }
2540     }
2541     \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2542     { \@@_make_preamble_S_i:n { #1 } }
2543     \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2544     {
2545         \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2546         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2547         \tl_gclear:N \g_@@_pre_cell_tl
2548         \tl_gput_right:Nn \g_@@_array_preamble_tl
2549         {
2550             > {
2551                 \@@_cell_begin:
2552                 \keys_set:nn { siunitx } { #1 }
2553                 \siunitx_cell_begin:w
2554             }
2555             c
2556             < { \siunitx_cell_end: \@@_cell_end: }
2557         }

```

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

```

2558     \int_gincr:N \c@jCol
2559     \@@_rec_preamble_after_col:n
2560 }

```

For (, [and \{.

```

2561 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2562 {
2563     \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.

```

2564     \int_if_zero:nTF \c@jCol
2565     {
2566         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2567         {

```

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

```

2568             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2569             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2570             \@@_rec_preamble:n #2
2571         }

```

```

2572     {
2573     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2574     \@@_make_preamble_iv:nn { #1 } { #2 }
2575     }
2576   }
2577   { \@@_make_preamble_iv:nn { #1 } { #2 } }
2578 }
2579 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2580 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2581 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2582 {
2583   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2584   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2585   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2586   {
2587     \@@_error:nn { delimiter~after~opening } { #2 }
2588     \@@_rec_preamble:n
2589   }
2590   { \@@_rec_preamble:n #2 }
2591 }

```

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

```

2592 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2593 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2594 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2595 {
2596   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2597   \tl_if_in:nnTF { ) ] \} } { #2 }
2598   { \@@_make_preamble_v:nnn #1 #2 }
2599   {
2600     \str_if_eq:nnTF { \@@_stop: } { #2 }
2601     {
2602       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2603       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2604       {
2605         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2606         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2607         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2608         \@@_rec_preamble:n #2
2609       }
2610     }
2611     {
2612       \tl_if_in:nnT { ( [ \{ \left } } { #2 }
2613       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2614       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2615       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2616       \@@_rec_preamble:n #2
2617     }
2618   }
2619 }
2620 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2621 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2622 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2623 {
2624   \str_if_eq:nnTF { \@@_stop: } { #3 }
2625   {
2626     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl

```

```

2627     {
2628     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2629     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2630     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2631     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2632     }
2633     {
2634     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2635     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2636     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2637     \@@_error:nn { double~closing~delimiter } { #2 }
2638     }
2639     }
2640     {
2641     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2642     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2643     \@@_error:nn { double~closing~delimiter } { #2 }
2644     \@@_rec_preamble:n #3
2645     }
2646     }

2647 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2648 { \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 @{...}.

```

2649 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2650 {
2651   \str_if_eq:nnTF { #1 } { < }
2652   \@@_rec_preamble_after_col_i:n
2653   {
2654     \str_if_eq:nnTF { #1 } { @ }
2655     \@@_rec_preamble_after_col_ii:n
2656     {
2657       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2658       {
2659         \tl_gput_right:Nn \g_@@_array_preamble_tl
2660         { ! { \skip_horizontal:N \arrayrulewidth } }
2661       }
2662       {
2663         \clist_if_in:NeT \l_@@_vlines_clist
2664         { \int_eval:n { \c@jCol + 1 } }
2665         {
2666           \tl_gput_right:Nn \g_@@_array_preamble_tl
2667           { ! { \skip_horizontal:N \arrayrulewidth } }
2668         }
2669       }
2670     }
2671   }
2672 }
2673 }

2674 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2675 {
2676   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2677   \@@_rec_preamble_after_col:n
2678 }

```

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.

```

2679 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2680 {

```

```

2681 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2682 {
2683   \tl_gput_right:Nn \g_@@_array_preamble_tl
2684   { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2685 }
2686 {
2687   \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2688   {
2689     \tl_gput_right:Nn \g_@@_array_preamble_tl
2690     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2691   }
2692   { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2693 }
2694 \@@_rec_preamble:n
2695 }

2696 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2697 {
2698   \tl_clear:N \l_tmpa_tl
2699   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2700   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2701 }

```

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.

```

2702 \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.

```

2703 \cs_new_protected:Npn \@@_X #1 #2
2704 {
2705   \str_if_eq:nnTF { #2 } { [ ]
2706     { \@@_make_preamble_X:w [ ] }
2707     { \@@_make_preamble_X:w [ ] #2 }
2708 }
2709 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2710 { \@@_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`).

```

2711 \keys_define:nn { nicematrix / X-column }
2712 { 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.

```

2713 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2714 {

```

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).

```

2715   \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).

```

2716   \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`.

```
2717 \int_zero_new:N \l_@@_weight_int
2718 \int_set_eq:NN \l_@@_weight_int \c_one_int
2719 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2720 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2721 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2722 {
2723   \@@_error_or_warning:n { negative-weight }
2724   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2725 }
2726 \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).

```
2727 \bool_if:NTF \l_@@_X_columns_aux_bool
2728 {
2729   \@@_make_preamble_ii_iv:nnn
2730   { \l_@@_weight_int \l_@@_X_columns_dim }
2731   { minipage }
2732   { \@@_no_update_width: }
2733 }
2734 {
2735   \tl_gput_right:Nn \g_@@_array_preamble_tl
2736   {
2737     > {
2738       \@@_cell_begin:
2739       \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`.

```
2740 \NotEmpty
```

The following code will nullify the box of the cell.

```
2741 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2742 { \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`.

```
2743 \begin { minipage } { 5 cm } \arraybackslash
2744 }
2745 c
2746 < {
2747   \end { minipage }
2748   \@@_cell_end:
2749 }
2750 }
2751 \int_gincr:N \c@jCol
2752 \@@_rec_preamble_after_col:n
2753 }
2754 }

2755 \cs_new_protected:Npn \@@_no_update_width:
2756 {
2757   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2758   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2759 }
```


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

```

2760 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2761 {
2762   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2763   { \int_eval:n { \c@jCol + 1 } }
2764   \tl_gput_right:Ne \g_@@_array_preamble_tl
2765   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2766   \@@_rec_preamble:n
2767 }

```

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.

```

2768 \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).

```

2769 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2770 { \@@_fatal:n { Preamble-forgotten } }
2771 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2772 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2773 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```

2774 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2775 {

```

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`.

```

2776   \multispan { #1 }
2777   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2778   \begingroup
2779   \bool_if:NT \c_@@_testphase_table_bool
2780   { \tbl_update_multicolumn_cell_data:n { #1 } }
2781   \cs_set_nopar:Npn \@addamp
2782   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preammerr 5 } }

```

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

```

2783   \tl_gclear:N \g_@@_preamble_tl
2784   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2785   \exp_args:No \@mkpream \g_@@_preamble_tl
2786   \@addtopreamble \@empty
2787   \endgroup
2788   \bool_if:NT \c_@@_recent_array_bool
2789   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2790   \int_compare:nNnT { #1 } > \c_one_int
2791   {
2792     \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2793     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2794     \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }

```

```

2795 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2796 {
2797   {
2798     \int_if_zero:nTF \c@jCol
2799     { \int_eval:n { \c@iRow + 1 } }
2800     { \int_use:N \c@iRow }
2801   }
2802   { \int_eval:n { \c@jCol + 1 } }
2803   {
2804     \int_if_zero:nTF \c@jCol
2805     { \int_eval:n { \c@iRow + 1 } }
2806     { \int_use:N \c@iRow }
2807   }
2808   { \int_eval:n { \c@jCol + #1 } }
2809   { } % for the name of the block
2810 }
2811 }

```

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

```

2812 \RenewDocumentCommand \cellcolor { 0 { } m }
2813 {
2814   \@@_test_color_inside:
2815   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2816   {
2817     \@@_rectanglecolor [ ##1 ]
2818     { \exp_not:n { ##2 } }
2819     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2820     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2821   }
2822   \ignorespaces
2823 }

```

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

```

2824 \cs_set_nopar:Npn \@sharp { #3 }
2825 \@arstrut
2826 \@preamble
2827 \null

```

We add some lines.

```

2828 \int_gadd:Nn \c@jCol { #1 - 1 }
2829 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2830 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2831 \ignorespaces
2832 }

```

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`.

```

2833 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2834 {
2835   \str_case:nnF { #1 }
2836   {
2837     c { \@@_make_m_preamble_i:n #1 }
2838     l { \@@_make_m_preamble_i:n #1 }
2839     r { \@@_make_m_preamble_i:n #1 }
2840     > { \@@_make_m_preamble_ii:nn #1 }
2841     ! { \@@_make_m_preamble_ii:nn #1 }
2842     @ { \@@_make_m_preamble_ii:nn #1 }
2843     | { \@@_make_m_preamble_iii:n #1 }
2844     p { \@@_make_m_preamble_iv:nnn t #1 }
2845     m { \@@_make_m_preamble_iv:nnn c #1 }
2846     b { \@@_make_m_preamble_iv:nnn b #1 }

```

```

2847     w { \@@_make_m_preamble_v:nnnn { } #1 }
2848     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2849     \q_stop { }
2850   }
2851   {
2852     \cs_if_exist:cTF { NC @ find @ #1 }
2853     {
2854       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2855       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2856     }
2857     {
2858       \str_if_eq:nnTF { #1 } { S }
2859       { \@@_fatal:n { unknown~column~type~S } }
2860       { \@@_fatal:nn { unknown~column~type } { #1 } }
2861     }
2862   }
2863 }

```

For c, l and r

```

2864 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2865 {
2866   \tl_gput_right:Nn \g_@@_preamble_tl
2867   {
2868     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2869     #1
2870     < \@@_cell_end:
2871   }

```

We test for the presence of a <.

```

2872   \@@_make_m_preamble_x:n
2873 }

```

For >, ! and @

```

2874 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2875 {
2876   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2877   \@@_make_m_preamble:n
2878 }

```

For l

```

2879 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2880 {
2881   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2882   \@@_make_m_preamble:n
2883 }

```

For p, m and b

```

2884 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2885 {
2886   \tl_gput_right:Nn \g_@@_preamble_tl
2887   {
2888     > {
2889       \@@_cell_begin:
2890       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2891       \mode_leave_vertical:
2892       \arraybackslash
2893       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2894     }
2895     c
2896     < {
2897       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2898       \end { minipage }
2899       \@@_cell_end:

```

```

2900     }
2901 }

```

We test for the presence of a <.

```

2902 \@@_make_m_preamble_x:n
2903 }

```

For w and W

```

2904 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2905 {
2906   \tl_gput_right:Nn \g_@@_preamble_tl
2907   {
2908     > {
2909       \dim_set:Nn \l_@@_col_width_dim { #4 }
2910       \hbox_set:Nw \l_@@_cell_box
2911       \@@_cell_begin:
2912       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2913     }
2914     c
2915     < {
2916       \@@_cell_end:
2917       \hbox_set_end:
2918       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2919       #1
2920       \@@_adjust_size_box:
2921       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2922     }
2923   }

```

We test for the presence of a <.

```

2924 \@@_make_m_preamble_x:n
2925 }

```

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

```

2926 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2927 {
2928   \str_if_eq:nnTF { #1 } { < }
2929   \@@_make_m_preamble_ix:n
2930   { \@@_make_m_preamble:n { #1 } }
2931 }
2932 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2933 {
2934   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2935   \@@_make_m_preamble_x:n
2936 }

```

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`).

```

2937 \cs_new_protected:Npn \@@_put_box_in_flow:
2938 {
2939   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2940   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2941   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2942   { \box_use_drop:N \l_tmpa_box }
2943   \@@_put_box_in_flow_i:
2944 }

```

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

```

2945 \cs_new_protected:Npn \@@_put_box_in_flow_i:

```

```

2946 {
2947   \pgfpicture
2948     \@@_qpoint:n { row - 1 }
2949     \dim_gset:eq:NN \g_tmpa_dim \pgf@y
2950     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2951     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2952     \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).

```

2953   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2954   {
2955     \int_set:Nn \l_tmpa_int
2956     {
2957       \str_range:Nnn
2958         \l_@@_baseline_tl
2959         6
2960         { \tl_count:o \l_@@_baseline_tl }
2961     }
2962     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2963   }
2964   {
2965     \str_if_eq:eeTF \l_@@_baseline_tl { t }
2966     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2967     {
2968       \str_if_eq:onTF \l_@@_baseline_tl { b }
2969       { \int_set_eq:NN \l_tmpa_int \c@iRow }
2970       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2971     }
2972     \bool_lazy_or:nnT
2973     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2974     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2975     {
2976       \@@_error:n { bad-value-for-baseline }
2977       \int_set_eq:NN \l_tmpa_int \c_one_int
2978     }
2979     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2980     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2981   }
2982   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2983   \endpgfpicture
2984   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2985   \box_use_drop:N \l_tmpa_box
2986 }

```

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).

```

2987 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2988 {

```

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.

```

2989   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2990   {
2991     \int_compare:nNnT \c@jCol > \c_one_int
2992     {
2993       \box_set_wd:Nn \l_@@_the_array_box
2994       { \box_wd:N \l_@@_the_array_box - \arraycolsep }

```

```

2995     }
2996 }

```

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

```

2997 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2998 \bool_if:NT \l_@@_caption_above_bool
2999 {
3000     \tl_if_empty:NF \l_@@_caption_tl
3001     {
3002         \bool_set_false:N \g_@@_caption_finished_bool
3003         \int_gzero:N \c@tabularnote
3004         \@@_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).

```

3005         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3006         {
3007             \tl_gput_right:Ne \g_@@_aux_tl
3008             {
3009                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3010                 { \int_use:N \g_@@_notes_caption_int }
3011             }
3012             \int_gzero:N \g_@@_notes_caption_int
3013         }
3014     }
3015 }

```

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

```

3016 \hbox
3017 {
3018     \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.

```

3019     \@@_create_extra_nodes:
3020     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3021 }

```

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

```

3022 \bool_lazy_any:nT
3023 {
3024     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3025     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3026     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3027 }
3028 \@@_insert_tabularnotes:
3029 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3030 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3031 \end { minipage }
3032 }

```

```

3033 \cs_new_protected:Npn \@@_insert_caption:
3034 {
3035     \tl_if_empty:NF \l_@@_caption_tl
3036     {
3037         \cs_if_exist:NTF \@captive

```

```

3038     { \@@_insert_caption_i: }
3039     { \@@_error:n { caption~outside~float } }
3040   }
3041 }

```

```

3042 \cs_new_protected:Npn \@@_insert_caption_i:
3043 {
3044   \group_begin:

```

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

```

3045   \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.

```

3046   \IfPackageLoadedT { floatrow }
3047   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3048   \tl_if_empty:NTF \l_@@_short_caption_tl
3049   { \caption }
3050   { \caption [ \l_@@_short_caption_tl ] }
3051   { \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.

```

3052   \bool_if:NF \g_@@_caption_finished_bool
3053   {
3054     \bool_gset_true:N \g_@@_caption_finished_bool
3055     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3056     \int_gzero:N \c@tabularnote
3057   }
3058   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3059   \group_end:
3060 }

```

```

3061 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3062 {
3063   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3064   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3065 }

```

```

3066 \cs_new_protected:Npn \@@_insert_tabularnotes:
3067 {
3068   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3069   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3070   \skip_vertical:N 0.65ex

```

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

```

3071   \group_begin:
3072   \l_@@_notes_code_before_tl
3073   \tl_if_empty:NF \g_@@_tabularnote_tl
3074   {
3075     \g_@@_tabularnote_tl \par
3076     \tl_gclear:N \g_@@_tabularnote_tl
3077   }

```

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.

```

3078   \int_compare:nNnT \c@tabularnote > \c_zero_int
3079   {
3080     \bool_if:NTF \l_@@_notes_para_bool
3081     {

```

```

3082     \begin { tabularnotes* }
3083         \seq_map_inline:Nn \g_@@_notes_seq
3084             { \@@_one_tabularnote:nn ##1 }
3085         \strut
3086     \end { tabularnotes* }

```

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

```

3087     \par
3088 }
3089 {
3090     \tabularnotes
3091         \seq_map_inline:Nn \g_@@_notes_seq
3092             { \@@_one_tabularnote:nn ##1 }
3093         \strut
3094     \endtabularnotes
3095 }
3096 }
3097 \unskip
3098 \group_end:
3099 \bool_if:NT \l_@@_notes_bottomrule_bool
3100 {
3101     \IfPackageLoadedTF { booktabs }
3102     {

```

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

```

3103         \skip_vertical:N \aboverulesep

```

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

```

3104         { \CT@arc@ \hrule height \heavyrulewidth }
3105     }
3106     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3107 }
3108 \l_@@_notes_code_after_tl
3109 \seq_gclear:N \g_@@_notes_seq
3110 \seq_gclear:N \g_@@_notes_in_caption_seq
3111 \int_gzero:N \c@tabularnote
3112 }

```

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

```

3113 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3114 {
3115     \tl_if_novalue:nTF { #1 }
3116     { \item }
3117     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3118 }

```

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`.

```

3119 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3120 {
3121     \pgfpicture
3122         \@@_qpoint:n { row - 1 }
3123         \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3124         \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3125         \dim_gsub:Nn \g_tmpa_dim \pgf@y
3126     \endpgfpicture
3127     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3128     \int_if_zero:nT \l_@@_first_row_int
3129     {

```



```

3130     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3131     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3132   }
3133   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3134 }

```

Now, the general case.

```

3135 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3136 {

```

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

```

3137   \str_if_eq:eeT \l_@@_baseline_tl { t }
3138   { \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`.

```

3139   \pgfpicture
3140   \@@_qpoint:n { row - 1 }
3141   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3142   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3143   {
3144     \int_set:Nn \l_tmpa_int
3145     {
3146       \str_range:Nnn
3147         \l_@@_baseline_tl
3148         6
3149         { \tl_count:o \l_@@_baseline_tl }
3150     }
3151     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3152   }
3153   {
3154     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3155     \bool_lazy_or:nnT
3156       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3157       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3158     {
3159       \@@_error:n { bad~value~for~baseline }
3160       \int_set:Nn \l_tmpa_int 1
3161     }
3162     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3163   }
3164   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3165   \endpgfpicture
3166   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3167   \int_if_zero:nT \l_@@_first_row_int
3168   {
3169     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3170     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3171   }
3172   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3173 }

```

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.

```

3174 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3175 {

```

We will compute the real width of both delimiters used.

```

3176   \dim_zero_new:N \l_@@_real_left_delim_dim
3177   \dim_zero_new:N \l_@@_real_right_delim_dim
3178   \hbox_set:Nn \l_tmpb_box
3179   {
3180     \c_math_toggle_token

```

```

3181     \left #1
3182     \vcenter
3183     {
3184         \vbox_to_ht:nn
3185         { \box_ht_plus_dp:N \l_tmpa_box }
3186         { }
3187     }
3188     \right .
3189     \c_math_toggle_token
3190 }
3191 \dim_set:Nn \l_@@_real_left_delim_dim
3192 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3193 \hbox_set:Nn \l_tmpb_box
3194 {
3195     \c_math_toggle_token
3196     \left .
3197     \vbox_to_ht:nn
3198     { \box_ht_plus_dp:N \l_tmpa_box }
3199     { }
3200     \right #2
3201     \c_math_toggle_token
3202 }
3203 \dim_set:Nn \l_@@_real_right_delim_dim
3204 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3205     \skip_horizontal:N \l_@@_left_delim_dim
3206     \skip_horizontal:N -\l_@@_real_left_delim_dim
3207     \@@_put_box_in_flow:
3208     \skip_horizontal:N \l_@@_right_delim_dim
3209     \skip_horizontal:N -\l_@@_real_right_delim_dim
3210 }

```

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).

```

3211 \NewDocumentEnvironment { @@-normal-syntax } { }

```

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

```

3212 {
3213     \peek_remove_spaces:n
3214     {
3215         \peek_meaning:NTF \end
3216         \@@_analyze_end:Nn
3217         {
3218             \@@_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`).

```

3219         \@@_array:o \g_@@_array_preamble_tl
3220     }
3221 }
3222 }
3223 {
3224     \@@_create_col_nodes:
3225     \endarray
3226 }

```

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

```
3227 \NewDocumentEnvironment { @@-light-syntax } { b }
3228 {
```

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`.

```
3229 \tl_if_empty:nT { #1 }
3230 { \@@_fatal:n { empty-environment } }
3231 \tl_if_in:nnT { #1 } { & }
3232 { \@@_fatal:n { ampersand-in~light-syntax } }
3233 \tl_if_in:nnT { #1 } { \ }
3234 { \@@_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`.

```
3235 \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3236 }
```

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.

```
3237 {
3238   \@@_create_col_nodes:
3239   \endarray
3240 }

3241 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3242 {
3243   \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).

```
3244 \seq_clear_new:N \l_@@_rows_seq
```

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

```
3245 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3246 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3247   \seq_set_split:Nee
3248   \seq_set_split:Non
3249   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3250 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3251 \tl_if_empty:NF \l_tmpa_tl
3252 { \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.

```
3253 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3254 { \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`).

```
3255 \tl_build_begin:N \l_@@_new_body_tl
3256 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3257 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3258 \@@_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).

```
3259 \seq_map_inline:Nn \l_@@_rows_seq
3260 {
3261   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3262   \@@_line_with_light_syntax:n { #1 }
3263 }
3264 \tl_build_end:N \l_@@_new_body_tl
3265 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3266 {
3267   \int_set:Nn \l_@@_last_col_int
3268   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3269 }
```

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.

```
3270 \@@_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`).

```
3271 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3272 }
3273 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3274 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3275 {
3276   \seq_clear_new:N \l_@@_cells_seq
3277   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3278   \int_set:Nn \l_@@_nb_cols_int
3279   {
3280     \int_max:nn
3281     \l_@@_nb_cols_int
3282     { \seq_count:N \l_@@_cells_seq }
3283   }
3284   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3285   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3286   \seq_map_inline:Nn \l_@@_cells_seq
3287   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3288 }
```

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`.

```
3289 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3290 {
3291   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3292   { \@@_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.

```
3293 \end { #2 }
3294 }
```

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`).

```
3295 \cs_new:Npn \@@_create_col_nodes:
3296 {
3297   \crr
```

```

3298 \int_if_zero:nT \l_@@_first_col_int
3299 {
3300   \omit
3301   \hbox_overlap_left:n
3302   {
3303     \bool_if:NT \l_@@_code_before_bool
3304     { \pgfsys@markposition { \@@_env: - col - 0 } }
3305     \pgfpicture
3306     \pgfrememberpicturepositiononpagetrue
3307     \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3308     \str_if_empty:NF \l_@@_name_str
3309     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3310     \endpgfpicture
3311     \skip_horizontal:N 2\col@sep
3312     \skip_horizontal:N \g_@@_width_first_col_dim
3313   }
3314   &
3315   }
3316   \omit

```

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

```

3317 \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`).

```

3318 \int_if_zero:nTF \l_@@_first_col_int
3319 {
3320   \bool_if:NT \l_@@_code_before_bool
3321   {
3322     \hbox
3323     {
3324       \skip_horizontal:N -0.5\arrayrulewidth
3325       \pgfsys@markposition { \@@_env: - col - 1 }
3326       \skip_horizontal:N 0.5\arrayrulewidth
3327     }
3328   }
3329   \pgfpicture
3330   \pgfrememberpicturepositiononpagetrue
3331   \pgfcoordinate { \@@_env: - col - 1 }
3332   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3333   \str_if_empty:NF \l_@@_name_str
3334   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3335   \endpgfpicture
3336 }
3337 {
3338   \bool_if:NT \l_@@_code_before_bool
3339   {
3340     \hbox
3341     {
3342       \skip_horizontal:N 0.5\arrayrulewidth
3343       \pgfsys@markposition { \@@_env: - col - 1 }
3344       \skip_horizontal:N -0.5\arrayrulewidth
3345     }
3346   }
3347   \pgfpicture
3348   \pgfrememberpicturepositiononpagetrue
3349   \pgfcoordinate { \@@_env: - col - 1 }
3350   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3351   \str_if_empty:NF \l_@@_name_str
3352   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3353   \endpgfpicture
3354 }

```

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.

```

3355 \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3356 \bool_if:NF \l_@@_auto_columns_width_bool
3357   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3358   {
3359     \bool_lazy_and:nnTF
3360       \l_@@_auto_columns_width_bool
3361       { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3362       { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3363       { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3364     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3365   }
3366 \skip_horizontal:N \g_tmpa_skip
3367 \hbox
3368   {
3369     \bool_if:NT \l_@@_code_before_bool
3370     {
3371       \hbox
3372       {
3373         \skip_horizontal:N -0.5\arrayrulewidth
3374         \pgfsys@markposition { \@@_env: - col - 2 }
3375         \skip_horizontal:N 0.5\arrayrulewidth
3376       }
3377     }
3378     \pgfpicture
3379     \pgfrememberpicturepositiononpagetrue
3380     \pgfcoordinate { \@@_env: - col - 2 }
3381     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3382     \str_if_empty:NF \l_@@_name_str
3383     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3384     \endpgfpicture
3385   }

```

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.

```

3386 \int_gset_eq:NN \g_tmpa_int \c_one_int
3387 \bool_if:NTF \g_@@_last_col_found_bool
3388   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3389   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3390   {
3391     &
3392     \omit
3393     \int_gincr:N \g_tmpa_int

```

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

```

3394     \skip_horizontal:N \g_tmpa_skip
3395     \bool_if:NT \l_@@_code_before_bool
3396     {
3397       \hbox
3398       {
3399         \skip_horizontal:N -0.5\arrayrulewidth
3400         \pgfsys@markposition
3401         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3402         \skip_horizontal:N 0.5\arrayrulewidth
3403       }
3404     }

```

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

```

3405     \pgfpicture
3406     \pgfrememberpicturepositiononpagetrue
3407     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }

```

```

3408     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3409 \str_if_empty:NF \l_@@_name_str
3410     {
3411       \pgfnodealias
3412         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3413         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3414     }
3415 \endpgfpicture
3416 }

3417 &
3418 \omit

```

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

```

3419 \int_if_zero:nT \g_@@_col_total_int
3420   { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3421 \skip_horizontal:N \g_tmpa_skip
3422 \int_gincr:N \g_tmpa_int
3423 \bool_lazy_any:nF
3424   {
3425     \g_@@_delims_bool
3426     \l_@@_tabular_bool
3427     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3428     \l_@@_exterior_arraycolsep_bool
3429     \l_@@_bar_at_end_of_pream_bool
3430   }
3431   { \skip_horizontal:N -\col@sep }
3432 \bool_if:NT \l_@@_code_before_bool
3433   {
3434     \hbox
3435     {
3436       \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.

```

3437     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3438       { \skip_horizontal:N -\arraycolsep }
3439 \pgfsys@markposition
3440   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3441 \skip_horizontal:N 0.5\arrayrulewidth
3442 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3443   { \skip_horizontal:N \arraycolsep }
3444   }
3445 }
3446 \pgfpicture
3447 \pgfrememberpicturepositiononpagetrue
3448 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3449   {
3450     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3451       {
3452         \pgfpoint
3453           { - 0.5 \arrayrulewidth - \arraycolsep }
3454           \c_zero_dim
3455       }
3456       { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3457   }
3458 \str_if_empty:NF \l_@@_name_str
3459   {
3460     \pgfnodealias
3461       { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3462       { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }

```

```

3463     }
3464 \endpgfpicture

3465 \bool_if:NT \g_@@_last_col_found_bool
3466 {
3467   \hbox_overlap_right:n
3468   {
3469     \skip_horizontal:N \g_@@_width_last_col_dim
3470     \skip_horizontal:N \col@sep
3471     \bool_if:NT \l_@@_code_before_bool
3472     {
3473       \pgfsys@markposition
3474       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3475     }
3476     \pgfpicture
3477     \pgfrememberpicturerepositiononpagetrue
3478     \pgfcoordinate
3479     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3480     \pgfpointorigin
3481     \str_if_empty:NF \l_@@_name_str
3482     {
3483       \pgfnodealias
3484       {
3485         \l_@@_name_str - col
3486         - \int_eval:n { \g_@@_col_total_int + 1 }
3487       }
3488       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3489     }
3490     \endpgfpicture
3491   }
3492 }
3493 % \cr
3494 }

```

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

```

3495 \tl_const:Nn \c_@@_preamble_first_col_tl
3496 {
3497   >
3498   {

```

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

```

3499     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3500     \bool_gset_true:N \g_@@_after_col_zero_bool
3501     \@@_begin_of_row:
3502     \hbox_set:Nw \l_@@_cell_box
3503     \@@_math_toggle:
3504     \@@_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”.

```

3505     \int_compare:nNnT \c@iRow > \c_zero_int
3506     {
3507       \bool_lazy_or:nnT
3508       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3509       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3510       {
3511         \l_@@_code_for_first_col_tl
3512         \xglobal \colorlet { nicematrix-first-col } { . }
3513       }
3514     }
3515 }

```


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`.

```

3516     l
3517     <
3518     {
3519         \@@_math_toggle:
3520         \hbox_set_end:
3521         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3522         \@@_adjust_size_box:
3523         \@@_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.

```

3524         \dim_gset:Nn \g_@@_width_first_col_dim
3525         { \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.

```

3526         \hbox_overlap_left:n
3527         {
3528             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3529             \@@_node_for_cell:
3530             { \box_use_drop:N \l_@@_cell_box }
3531             \skip_horizontal:N \l_@@_left_delim_dim
3532             \skip_horizontal:N \l_@@_left_margin_dim
3533             \skip_horizontal:N \l_@@_extra_left_margin_dim
3534         }
3535         \bool_gset_false:N \g_@@_empty_cell_bool
3536         \skip_horizontal:N -2\col@sep
3537     }
3538 }

```

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

```

3539 \tl_const:Nn \c_@@_preamble_last_col_tl
3540 {
3541     >
3542     {
3543         \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).

```

3544         \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.

```

3545         \bool_gset_true:N \g_@@_last_col_found_bool
3546         \int_gincr:N \c@jCol
3547         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3548         \hbox_set:Nw \l_@@_cell_box
3549         \@@_math_toggle:
3550         \@@_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”.

```

3551         \int_compare:nNnT \c@iRow > \c_zero_int
3552         {
3553             \bool_lazy_or:nnT
3554             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3555             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3556             {
3557                 \l_@@_code_for_last_col_tl
3558                 \xglobal \colorlet { nicematrix-last-col } { . }
3559             }
3560         }
3561     }
3562     l
3563     <

```

```

3564 {
3565   \@@_math_toggle:
3566   \hbox_set_end:
3567   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3568   \@@_adjust_size_box:
3569   \@@_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.

```

3570   \dim_gset:Nn \g_@@_width_last_col_dim
3571   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3572   \skip_horizontal:N -2\col@sep

```

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

```

3573   \hbox_overlap_right:n
3574   {
3575     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3576     {
3577       \skip_horizontal:N \l_@@_right_delim_dim
3578       \skip_horizontal:N \l_@@_right_margin_dim
3579       \skip_horizontal:N \l_@@_extra_right_margin_dim
3580       \@@_node_for_cell:
3581     }
3582   }
3583   \bool_gset_false:N \g_@@_empty_cell_bool
3584 }
3585 }

```

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

```

3586 \NewDocumentEnvironment { NiceArray } { }
3587 {
3588   \bool_gset_false:N \g_@@_delims_bool
3589   \str_if_empty:NT \g_@@_name_env_str
3590   { \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).

```

3591   \NiceArrayWithDelims . .
3592 }
3593 { \endNiceArrayWithDelims }

```

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

```

3594 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3595 {
3596   \NewDocumentEnvironment { #1 NiceArray } { }
3597   {
3598     \bool_gset_true:N \g_@@_delims_bool
3599     \str_if_empty:NT \g_@@_name_env_str
3600     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3601     \@@_test_if_math_mode:
3602     \NiceArrayWithDelims #2 #3
3603   }
3604   { \endNiceArrayWithDelims }
3605 }
3606 \@@_def_env:nnn p ( )
3607 \@@_def_env:nnn b [ ]
3608 \@@_def_env:nnn B \{ \}
3609 \@@_def_env:nnn v | |
3610 \@@_def_env:nnn V \| \|

```

13 The environment `{NiceMatrix}` and its variants

```

3611 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3612 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3613 {
3614   \bool_set_false:N \l_@@_preamble_bool
3615   \tl_clear:N \l_tmpa_tl
3616   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3617     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3618   \tl_put_right:Nn \l_tmpa_tl
3619     {
3620       *
3621       {
3622         \int_case:nnF \l_@@_last_col_int
3623           {
3624             { -2 } { \c@MaxMatrixCols }
3625             { -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).

```

3626       }
3627       { \int_eval:n { \l_@@_last_col_int - 1 } }
3628     }
3629     { #2 }
3630   }
3631   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3632   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3633 }
3634 \clist_map_inline:nn { p , b , B , v , V }
3635 {
3636   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3637   {
3638     \bool_gset_true:N \g_@@_delims_bool
3639     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3640     \int_if_zero:nT \l_@@_last_col_int
3641     {
3642       \bool_set_true:N \l_@@_last_col_without_value_bool
3643       \int_set:Nn \l_@@_last_col_int { -1 }
3644     }
3645     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3646     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3647   }
3648   { \use:c { end #1 NiceArray } }
3649 }

```

We define also an environment `{NiceMatrix}`

```

3650 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3651 {
3652   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3653   \int_if_zero:nT \l_@@_last_col_int
3654   {
3655     \bool_set_true:N \l_@@_last_col_without_value_bool
3656     \int_set:Nn \l_@@_last_col_int { -1 }
3657   }
3658   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3659   \bool_lazy_or:nnT
3660     { \clist_if_empty_p:N \l_@@_vlines_clist }
3661     { \l_@@_except_borders_bool }
3662     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3663   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3664 }
3665 { \endNiceArray }

```

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

```
3666 \cs_new_protected:Npn \@@_NotEmpty:
3667   { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3668 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3669   {
```

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`.

```
3670     \dim_compare:nNt \l_@@_width_dim = \c_zero_dim
3671     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3672     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3673     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3674     \tl_if_empty:NF \l_@@_short_caption_tl
3675     {
3676       \tl_if_empty:NT \l_@@_caption_tl
3677       {
3678         \@@_error_or_warning:n { short-caption-without-caption }
3679         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3680       }
3681     }
3682     \tl_if_empty:NF \l_@@_label_tl
3683     {
3684       \tl_if_empty:NT \l_@@_caption_tl
3685       { \@@_error_or_warning:n { label-without-caption } }
3686     }
3687     \NewDocumentEnvironment { TabularNote } { b }
3688     {
3689       \bool_if:NTF \l_@@_in_code_after_bool
3690       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3691       {
3692         \tl_if_empty:NF \g_@@_tabularnote_tl
3693         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3694         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3695       }
3696     }
3697     { }
3698     \@@_settings_for_tabular:
3699     \NiceArray { #2 }
3700   }
3701   {
3702     \endNiceArray
3703     \bool_if:NT \c_@@_recent_array_bool
3704     { \UseTaggingSocket { tbl / hmode / end } }
3705   }
3706 \cs_new_protected:Npn \@@_settings_for_tabular:
3707   {
3708     \bool_set_true:N \l_@@_tabular_bool
3709     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3710     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3711     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3712   }

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

```

3720 \NiceArray { #3 }
3721 }
3722 {
3723 \endNiceArray
3724 \int_if_zero:nT \g_@@_total_X_weight_int
3725 { \@@_error:n { NiceTabularX~without~X } }
3726 }

3727 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3728 {
3729 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3730 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3731 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3732 \@@_settings_for_tabular:
3733 \NiceArray { #3 }
3734 }
3735 { \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`).

```

3736 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3737 {
3738 \bool_lazy_all:nT
3739 {
3740 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3741 \l_@@_hvlines_bool
3742 { ! \g_@@_delims_bool }
3743 { ! \l_@@_except_borders_bool }
3744 }
3745 {
3746 \bool_set_true:N \l_@@_except_borders_bool
3747 \clist_if_empty:NF \l_@@_corners_clist
3748 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3749 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3750 {
3751 \@@_stroke_block:nnn
3752 {
3753 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3754 draw = \l_@@_rules_color_tl
3755 }
3756 { 1-1 }
3757 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3758 }
3759 }
3760 }

3761 \cs_new_protected:Npn \@@_after_array:
3762 {

```

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`.

```

3763 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3764 \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.

```
3765 \bool_if:NT \g_@@_last_col_found_bool
3766 { \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`.

```
3767 \bool_if:NT \l_@@_last_col_without_value_bool
3768 { \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.

```
3769 \bool_if:NT \l_@@_last_row_without_value_bool
3770 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3771 \tl_gput_right:Ne \g_@@_aux_tl
3772 {
3773   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3774   {
3775     \int_use:N \l_@@_first_row_int ,
3776     \int_use:N \c@iRow ,
3777     \int_use:N \g_@@_row_total_int ,
3778     \int_use:N \l_@@_first_col_int ,
3779     \int_use:N \c@jCol ,
3780     \int_use:N \g_@@_col_total_int
3781   }
3782 }
```

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`).

```
3783 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3784 {
3785   \tl_gput_right:Ne \g_@@_aux_tl
3786   {
3787     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3788     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3789   }
3790 }
3791 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3792 {
3793   \tl_gput_right:Ne \g_@@_aux_tl
3794   {
3795     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3796     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3797     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3798     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3799   }
3800 }
```

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

```
3801 \@@_create_diag_nodes:
```

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

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

```

3808     }
3809     \int_step_inline:nn \c@jCol
3810     {
3811         \pgfnodealias
3812         { \l_@@_env: - last - ##1 }
3813         { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3814     }
3815     \str_if_empty:NF \l_@@_name_str
3816     {
3817         \int_step_inline:nn \c@iRow
3818         {
3819             \pgfnodealias
3820             { \l_@@_name_str - ##1 - last }
3821             { \l_@@_env: - ##1 - \int_use:N \c@jCol }
3822         }
3823         \int_step_inline:nn \c@jCol
3824         {
3825             \pgfnodealias
3826             { \l_@@_name_str - last - ##1 }
3827             { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3828         }
3829     }
3830     \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.

```

3831     \bool_if:NT \l_@@_parallelize_diags_bool
3832     {
3833         \int_gzero_new:N \g_@@_ddots_int
3834         \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.

```

3835         \dim_gzero_new:N \g_@@_delta_x_one_dim
3836         \dim_gzero_new:N \g_@@_delta_y_one_dim
3837         \dim_gzero_new:N \g_@@_delta_x_two_dim
3838         \dim_gzero_new:N \g_@@_delta_y_two_dim
3839     }

3840     \int_zero_new:N \l_@@_initial_i_int
3841     \int_zero_new:N \l_@@_initial_j_int
3842     \int_zero_new:N \l_@@_final_i_int
3843     \int_zero_new:N \l_@@_final_j_int
3844     \bool_set_false:N \l_@@_initial_open_bool
3845     \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.

```

3846     \bool_if:NT \l_@@_small_bool
3847     {
3848         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3849         \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.

```

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

```

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

```

3853     { 0.6 \l_@@_xdots_shorten_end_dim }
3854 }

```

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

```

3855 \@@_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.

```

3856 \clist_if_empty:NF \l_@@_corners_clist \@@_compute_corners:

```

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

```

3857 \@@_adjust_pos_of_blocks_seq:
3858 \@@_deal_with_rounded_corners:
3859 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3860 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3861 \IfPackageLoadedT { tikz }
3862 {
3863   \tikzset
3864   {
3865     every-picture / .style =
3866     {
3867       overlay ,
3868       remember~picture ,
3869       name~prefix = \@@_env: -
3870     }
3871   }
3872 }
3873 \bool_if:NT \c_@@_recent_array_bool
3874 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3875 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3876 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3877 \cs_set_eq:NN \OverBrace \@@_OverBrace
3878 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3879 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3880 \cs_set_eq:NN \line \@@_line
3881 \g_@@_pre_code_after_tl
3882 \tl_gclear:N \g_@@_pre_code_after_tl

```

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

```

3883 \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).

```

3884 \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`).

```

3885 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3886 { \@@_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:`.

```

3887   \bool_set_true:N \l_@@_in_code_after_bool
3888   \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3889   \scan_stop:
3890   \tl_gclear:N \g_nicematrix_code_after_tl
3891   \group_end:

```

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

```

3892   \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3893   \tl_if_empty:NF \g_@@_pre_code_before_tl
3894   {
3895     \tl_gput_right:Ne \g_@@_aux_tl
3896     {
3897       \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3898       { \exp_not:o \g_@@_pre_code_before_tl }
3899     }
3900     \tl_gclear:N \g_@@_pre_code_before_tl
3901   }
3902   \tl_if_empty:NF \g_nicematrix_code_before_tl
3903   {
3904     \tl_gput_right:Ne \g_@@_aux_tl
3905     {
3906       \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3907       { \exp_not:o \g_nicematrix_code_before_tl }
3908     }
3909     \tl_gclear:N \g_nicematrix_code_before_tl
3910   }

3911   \str_gclear:N \g_@@_name_env_str
3912   \@@_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`.

```

3913   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3914   }

```

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`.

```

3915   \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3916   { \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).

```

3917   \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3918   {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3919   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3920     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3921 }

```

The following command must *not* be protected.

```

3922 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3923 {
3924   { #1 }
3925   { #2 }
3926   {
3927     \int_compare:nNnTF { #3 } > { 99 }
3928     { \int_use:N \c@iRow }
3929     { #3 }
3930   }
3931   {
3932     \int_compare:nNnTF { #4 } > { 99 }
3933     { \int_use:N \c@jCol }
3934     { #4 }
3935   }
3936   { #5 }
3937 }

```

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).

```

3938 \hook_gput_code:nnn { begindocument } { . }
3939 {
3940   \cs_new_protected:Npe \@@_draw_dotted_lines:
3941     {
3942       \c_@@_pgfortikzpicture_tl
3943       \@@_draw_dotted_lines_i:
3944       \c_@@_endpgfortikzpicture_tl
3945     }
3946 }

```

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

```

3947 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3948 {
3949   \pgfrememberpicturepositiononpagetrue
3950   \pgf@relevantforpicturesizefalse
3951   \g_@@_HVdotsfor_lines_tl
3952   \g_@@_Vdots_lines_tl
3953   \g_@@_Ddots_lines_tl
3954   \g_@@_Iddots_lines_tl
3955   \g_@@_Cdots_lines_tl
3956   \g_@@_Ldots_lines_tl
3957 }

3958 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3959 {
3960   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3961   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3962 }

```

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

```

3963 \pgfdeclareshape { @@_diag_node }
3964 {
3965   \savedanchor { \five }
3966   {
3967     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

3968     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3969   }
3970   \anchor { 5 } { \five }
3971   \anchor { center } { \pgfpointorigin }
3972   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3973   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3974   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3975   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3976   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
3977   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3978   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3979   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3980   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3981   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3982 }

```

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).

```

3983 \cs_new_protected:Npn \@@_create_diag_nodes:
3984 {
3985   \pgfpicture
3986   \pgfrememberpicturepositiononpagetrue
3987   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3988   {
3989     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3990     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3991     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3992     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3993     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3994     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3995     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3996     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3997     \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.

```

3998     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3999     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4000     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4001     \str_if_empty:NF \l_@@_name_str
4002     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4003   }

```

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

```

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

```


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.

```

4037     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4038     \if_int_compare:w #3 = \c_one_int
4039     \bool_set_true:N \l_@@_final_open_bool
4040     \else:
4041     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4042     \bool_set_true:N \l_@@_final_open_bool
4043     \fi:
4044     \fi:
4045     \else:
4046     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4047     \if_int_compare:w #4 = -1
4048     \bool_set_true:N \l_@@_final_open_bool
4049     \fi:
4050     \else:
4051     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4052     \if_int_compare:w #4 = \c_one_int
4053     \bool_set_true:N \l_@@_final_open_bool
4054     \fi:
4055     \fi:
4056     \fi:
4057     \fi:
4058     \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.

```

4059     {

```

We do a step backwards.

```

4060     \int_sub:Nn \l_@@_final_i_int { #3 }
4061     \int_sub:Nn \l_@@_final_j_int { #4 }
4062     \bool_set_true:N \l_@@_stop_loop_bool
4063     }

```

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`.

```

4064     {
4065     \cs_if_exist:cTF
4066     {
4067     @@ _ dotted _
4068     \int_use:N \l_@@_final_i_int -
4069     \int_use:N \l_@@_final_j_int
4070     }
4071     {
4072     \int_sub:Nn \l_@@_final_i_int { #3 }
4073     \int_sub:Nn \l_@@_final_j_int { #4 }
4074     \bool_set_true:N \l_@@_final_open_bool
4075     \bool_set_true:N \l_@@_stop_loop_bool
4076     }
4077     {
4078     \cs_if_exist:cTF
4079     {
4080     pgf @ sh @ ns @ \@@_env:
4081     - \int_use:N \l_@@_final_i_int
4082     - \int_use:N \l_@@_final_j_int
4083     }
4084     { \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.

```

4085     {

```

```

4086         \cs_set_nopar:cpn
4087         {
4088             @@ _ dotted _
4089             \int_use:N \l_@@_final_i_int -
4090             \int_use:N \l_@@_final_j_int
4091         }
4092         { }
4093     }
4094 }
4095 }
4096 }

```

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

```

4097     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4098     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4099     \bool_do_until:Nn \l_@@_stop_loop_bool
4100     {
4101         \int_sub:Nn \l_@@_initial_i_int { #3 }
4102         \int_sub:Nn \l_@@_initial_j_int { #4 }
4103         \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.

```

4104         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4105         \if_int_compare:w #3 = \c_one_int
4106             \bool_set_true:N \l_@@_initial_open_bool
4107         \else:

```

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

```

4108             \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4109                 \bool_set_true:N \l_@@_initial_open_bool
4110             \fi:
4111         \fi:
4112     \else:
4113         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4114             \if_int_compare:w #4 = \c_one_int
4115                 \bool_set_true:N \l_@@_initial_open_bool
4116             \fi:
4117         \else:
4118             \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4119                 \if_int_compare:w #4 = -1
4120                     \bool_set_true:N \l_@@_initial_open_bool
4121                 \fi:
4122             \fi:
4123         \fi:
4124     \fi:
4125     \bool_if:NTF \l_@@_initial_open_bool
4126     {
4127         \int_add:Nn \l_@@_initial_i_int { #3 }
4128         \int_add:Nn \l_@@_initial_j_int { #4 }
4129         \bool_set_true:N \l_@@_stop_loop_bool
4130     }
4131     {
4132         \cs_if_exist:cTF
4133         {
4134             @@ _ dotted _
4135             \int_use:N \l_@@_initial_i_int -
4136             \int_use:N \l_@@_initial_j_int
4137         }

```

```

4138     {
4139         \int_add:Nn \l_@@_initial_i_int { #3 }
4140         \int_add:Nn \l_@@_initial_j_int { #4 }
4141         \bool_set_true:N \l_@@_initial_open_bool
4142         \bool_set_true:N \l_@@_stop_loop_bool
4143     }
4144     {
4145         \cs_if_exist:cTF
4146         {
4147             pgf @ sh @ ns @ \@@_env:
4148             - \int_use:N \l_@@_initial_i_int
4149             - \int_use:N \l_@@_initial_j_int
4150         }
4151         { \bool_set_true:N \l_@@_stop_loop_bool }
4152         {
4153             \cs_set_nopar:cpn
4154             {
4155                 @@ _ dotted _
4156                 \int_use:N \l_@@_initial_i_int -
4157                 \int_use:N \l_@@_initial_j_int
4158             }
4159             { }
4160         }
4161     }
4162 }
4163 }

```

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.

```

4164     \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4165     {
4166         { \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`.

```

4167         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4168         { \int_use:N \l_@@_final_i_int }
4169         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4170         { } % for the name of the block
4171     }
4172 }

```

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.

```

4173 \cs_new_protected:Npn \@@_open_shorten:
4174 {
4175     \bool_if:NT \l_@@_initial_open_bool
4176     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4177     \bool_if:NT \l_@@_final_open_bool
4178     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4179 }

```

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).

```

4180 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4181 {
4182     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4183 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4184 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4185 \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`.

```

4186 \seq_if_empty:NF \g_@@_submatrix_seq
4187 {
4188   \seq_map_inline:Nn \g_@@_submatrix_seq
4189   { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4190 }
4191 }

```

#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.

```

4192 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnn #1 #2 #3 #4 #5 #6
4193 {
4194   \if_int_compare:w #3 > #1
4195   \else:
4196     \if_int_compare:w #1 > #5
4197     \else:
4198       \if_int_compare:w #4 > #2
4199       \else:
4200         \if_int_compare:w #2 > #6
4201         \else:
4202           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4203           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4204           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4205           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4206         \fi:
4207       \fi:
4208     \fi:
4209   \fi:
4210 }

4211 \cs_new_protected:Npn \@@_set_initial_coords:
4212 {
4213   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4214   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4215 }
4216 \cs_new_protected:Npn \@@_set_final_coords:
4217 {

```



```

4218     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4219     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4220   }
4221   \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4222   {
4223     \pgfpointanchor
4224     {
4225       \@@_env:
4226       - \int_use:N \l_@@_initial_i_int
4227       - \int_use:N \l_@@_initial_j_int
4228     }
4229     { #1 }
4230     \@@_set_initial_coords:
4231   }
4232   \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4233   {
4234     \pgfpointanchor
4235     {
4236       \@@_env:
4237       - \int_use:N \l_@@_final_i_int
4238       - \int_use:N \l_@@_final_j_int
4239     }
4240     { #1 }
4241     \@@_set_final_coords:
4242   }
4243   \cs_new_protected:Npn \@@_open_x_initial_dim:
4244   {
4245     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4246     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4247     {
4248       \cs_if_exist:cT
4249       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4250       {
4251         \pgfpointanchor
4252         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4253         { west }
4254         \dim_set:Nn \l_@@_x_initial_dim
4255         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4256       }
4257     }

```

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

```

4258     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4259     {
4260       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4261       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4262       \dim_add:Nn \l_@@_x_initial_dim \col@sep
4263     }
4264   }
4265   \cs_new_protected:Npn \@@_open_x_final_dim:
4266   {
4267     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4268     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4269     {
4270       \cs_if_exist:cT
4271       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4272       {
4273         \pgfpointanchor
4274         { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4275         { east }
4276         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4277         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4278       }

```

```
4279     }
```

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

```
4280     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4281     {
4282         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4283         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4284         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4285     }
4286 }
```

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.

```
4287 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4288 {
4289     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4290     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4291     {
4292         \@@_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.

```
4293         \group_begin:
4294         \@@_open_shorten:
4295         \int_if_zero:nTF { #1 }
4296         { \color { nicematrix-first-row } }
4297         {
```

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.

```
4298             \int_compare:nNnT { #1 } = \l_@@_last_row_int
4299             { \color { nicematrix-last-row } }
4300         }
4301         \keys_set:nn { nicematrix / xdots } { #3 }
4302         \@@_color:o \l_@@_xdots_color_tl
4303         \@@_actually_draw_Ldots:
4304     \group_end:
4305 }
4306 }
```

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`.

```
4307 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4308 {
4309     \bool_if:NTF \l_@@_initial_open_bool
4310     {
4311         \@@_open_x_initial_dim:
4312         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4313         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4314     }
4315     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4316 \bool_if:NTF \l_@@_final_open_bool
4317 {
4318   \@@_open_x_final_dim:
4319   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4320   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4321 }
4322 { \@@_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.

```

4323 \bool_lazy_all:nTF
4324 {
4325   \l_@@_initial_open_bool
4326   \l_@@_final_open_bool
4327   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4328 }
4329 {
4330   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4331   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4332 }

```

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 (?).

```

4333 {
4334   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4335   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4336 }
4337 \@@_draw_line:
4338 }

```

The first and the second arguments are the coordinates of the cell where the command has been issued. The third argument is the list of the options.

```

4339 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4340 {
4341   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4342   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4343   {
4344     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1

```

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

```

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

```

We remind that, when there is a “last row” `\l_@@_last_row_int` will always be (after the construction of the array) the number of that “last row” even if the option `last-row` has been used without value.

```

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

```

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.

```

4359 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4360 {
4361   \bool_if:NTF \l_@@_initial_open_bool
4362     { \@@_open_x_initial_dim: }
4363     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4364   \bool_if:NTF \l_@@_final_open_bool
4365     { \@@_open_x_final_dim: }
4366     { \@@_set_final_coords_from_anchor:n { mid-west } }
4367   \bool_lazy_and:mnTF
4368     \l_@@_initial_open_bool
4369     \l_@@_final_open_bool
4370   {
4371     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4372     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4373     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4374     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4375     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4376   }
4377   {
4378     \bool_if:NT \l_@@_initial_open_bool
4379       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4380     \bool_if:NT \l_@@_final_open_bool
4381       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4382   }
4383   \@@_draw_line:
4384 }

4385 \cs_new_protected:Npn \@@_open_y_initial_dim:
4386 {
4387   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4388   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4389     {
4390       \cs_if_exist:cT
4391         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4392         {
4393           \pgfpointanchor
4394             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4395             { north }
4396           \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4397             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4398         }
4399     }
4400   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4401     {
4402       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4403       \dim_set:Nn \l_@@_y_initial_dim
4404         {
4405           \fp_to_dim:n
4406             {
4407               \pgf@y
4408               + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4409             }
4410         }
4411     }
4412 }

```

```

4413 \cs_new_protected:Npn \l_@@_open_y_final_dim:
4414 {
4415   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4416   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4417   {
4418     \cs_if_exist:cT
4419     { \pgf @ sh @ ns @ \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4420     {
4421       \pgfpointanchor
4422       { \l_@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4423       { south }
4424       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4425       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4426     }
4427   }
4428   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4429   {
4430     \l_@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4431     \dim_set:Nn \l_@@_y_final_dim
4432     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4433   }
4434 }

```

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.

```

4435 \cs_new_protected:Npn \l_@@_draw_Vdots:nnn #1 #2 #3
4436 {
4437   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4438   \cs_if_free:cT { \l_@@_dotted_#1 - #2 }
4439   {
4440     \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.

```

4441   \group_begin:
4442   \l_@@_open_shorten:
4443   \int_if_zero:nTF { #2 }
4444   { \color { nicematrix-first-col } }
4445   {
4446     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4447     { \color { nicematrix-last-col } }
4448   }
4449   \keys_set:nn { nicematrix / xdots } { #3 }
4450   \l_@@_color:o \l_@@_xdots_color_tl
4451   \l_@@_actually_draw_Vdots:
4452   \group_end:
4453 }
4454 }

```

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`.

```

4455 \cs_new_protected:Npn \l_@@_actually_draw_Vdots:
4456 {

```

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

```
4457 \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.

```
4458 {
4459   \@@_open_y_initial_dim:
4460   \@@_open_y_final_dim:
4461   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4462 {
4463   \@@_qpoint:n { col - 1 }
4464   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4465   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4466   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4467   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4468 }
4469 {
4470   \bool_lazy_and:nnTF
4471   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4472   { \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”.

```
4473 {
4474   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4475   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4476   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4477   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4478   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4479 }
```

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

```
4480 {
4481   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4482   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4483   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4484   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4485 }
4486 }
4487 }
```

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.

```
4488 {
4489   \bool_set_false:N \l_tmpa_bool
4490   \bool_if:NF \l_@@_initial_open_bool
4491   {
4492     \bool_if:NF \l_@@_final_open_bool
4493     {
4494       \@@_set_initial_coords_from_anchor:n { south-west }
4495       \@@_set_final_coords_from_anchor:n { north-west }
4496       \bool_set:Nn \l_tmpa_bool
4497       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4498     }
4499   }
```

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

```
4500 \bool_if:NTF \l_@@_initial_open_bool
4501 {
4502   \@@_open_y_initial_dim:
4503   \@@_set_final_coords_from_anchor:n { north }
4504   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4505 }
4506 {
4507   \@@_set_initial_coords_from_anchor:n { south }
4508   \bool_if:NTF \l_@@_final_open_bool
```

```
4509         \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.

```
4510         {
4511         \l_@@_set_final_coords_from_anchor:n { north }
4512         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4513         {
4514         \dim_set:Nn \l_@@_x_initial_dim
4515         {
4516         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4517         \l_@@_x_initial_dim \l_@@_x_final_dim
4518         }
4519         }
4520         }
4521     }
4522 }
4523 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4524 \l_@@_draw_line:
4525 }
```

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.

```
4526 \cs_new_protected:Npn \l_@@_draw_Ddots:nnn #1 #2 #3
4527 {
4528     \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4529     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4530     {
4531         \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.

```
4532     \group_begin:
4533     \l_@@_open_shorten:
4534     \keys_set:nn { nicematrix / xdots } { #3 }
4535     \l_@@_color:o \l_@@_xdots_color_tl
4536     \l_@@_actually_draw_Ddots:
4537     \group_end:
4538     }
4539 }
```

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`.

```
4540 \cs_new_protected:Npn \l_@@_actually_draw_Ddots:
4541 {
4542     \bool_if:NTF \l_@@_initial_open_bool
4543     {
4544         \l_@@_open_y_initial_dim:
4545         \l_@@_open_x_initial_dim:
```

```

4546     }
4547     { \@@_set_initial_coords_from_anchor:n { south-east } }
4548 \bool_if:NTF \l_@@_final_open_bool
4549     {
4550     \@@_open_x_final_dim:
4551     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4552     }
4553     { \@@_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.

```

4554     \bool_if:NT \l_@@_parallelize_diags_bool
4555     {
4556     \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).

```

4557     \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.

```

4558     {
4559     \dim_gset:Nn \g_@@_delta_x_one_dim
4560     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4561     \dim_gset:Nn \g_@@_delta_y_one_dim
4562     { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4563     }

```

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`.

```

4564     {
4565     \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4566     {
4567     \dim_set:Nn \l_@@_y_final_dim
4568     {
4569     \l_@@_y_initial_dim +
4570     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4571     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4572     }
4573     }
4574     }
4575     }
4576 \@@_draw_line:
4577 }

```

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.

```

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

```

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

```


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.`

```

4592 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4593 {
4594   \bool_if:NTF \l_@@_initial_open_bool
4595     {
4596       \@@_open_y_initial_dim:
4597       \@@_open_x_initial_dim:
4598     }
4599     { \@@_set_initial_coords_from_anchor:n { south-west } }
4600   \bool_if:NTF \l_@@_final_open_bool
4601     {
4602       \@@_open_y_final_dim:
4603       \@@_open_x_final_dim:
4604     }
4605     { \@@_set_final_coords_from_anchor:n { north-east } }
4606   \bool_if:NT \l_@@_parallelize_diags_bool
4607     {
4608       \int_gincr:N \g_@@_iddots_int
4609       \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4610         {
4611           \dim_gset:Nn \g_@@_delta_x_two_dim
4612             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4613           \dim_gset:Nn \g_@@_delta_y_two_dim
4614             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4615         }
4616         {
4617           \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4618             {
4619               \dim_set:Nn \l_@@_y_final_dim
4620                 {
4621                   \l_@@_y_initial_dim +
4622                   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4623                   \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4624                 }
4625             }
4626         }
4627     }
4628   \@@_draw_line:
4629 }

```

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`

```

4630 \cs_new_protected:Npn \@@_draw_line:
4631 {
4632   \pgfrememberpicturepositiononpagetrue
4633   \pgf@relevantforpicturesizefalse
4634   \bool_lazy_or:nnTF
4635     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4636     \l_@@_dotted_bool
4637     \@@_draw_standard_dotted_line:
4638     \@@_draw_unstandard_dotted_line:
4639 }

```

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.

```

4640 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4641 {
4642   \begin { scope }
4643     \@@_draw_unstandard_dotted_line:o
4644     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4645 }

```

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.

```

4646 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4647 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4648 {
4649   \@@_draw_unstandard_dotted_line:nooo
4650   { #1 }
4651   \l_@@_xdots_up_tl
4652   \l_@@_xdots_down_tl
4653   \l_@@_xdots_middle_tl
4654 }

```

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

```

4655 \hook_gput_code:nnn { begindocument } { . }
4656 {
4657   \IfPackageLoadedT { tikz }
4658   {
4659     \tikzset
4660     {
4661       @@_node_above / .style = { sloped , above } ,
4662       @@_node_below / .style = { sloped , below } ,
4663       @@_node_middle / .style =
4664       {
4665         sloped ,
4666         inner~sep = \c_@@_innersep_middle_dim
4667       }
4668     }
4669   }
4670 }

```

```

4671 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nmmn { n o o o }
4672 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nmmn #1 #2 #3 #4
4673 {

```

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.

```

4674 \dim_zero_new:N \l_@@_l_dim
4675 \dim_set:Nn \l_@@_l_dim
4676 {
4677   \fp_to_dim:n
4678   {
4679     sqrt
4680     (
4681       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4682       +
4683       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4684     )
4685   }
4686 }

```

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.

```

4687 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4688 {
4689   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4690   \@@_draw_unstandard_dotted_line_i:
4691 }

```

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

```

4692 \bool_if:NT \l_@@_xdots_h_labels_bool
4693 {
4694   \tikzset
4695   {
4696     @@_node_above / .style = { auto = left } ,
4697     @@_node_below / .style = { auto = right } ,
4698     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4699   }
4700 }
4701 \tl_if_empty:nF { #4 }
4702 { \tikzset { @@_node_middle / .append-style = { fill = white } } }
4703 \draw
4704 [ #1 ]
4705 ( \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).

```

4706 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4707 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4708 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4709 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4710 \end { scope }
4711 }
4712 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4713 {
4714   \dim_set:Nn \l_tmpa_dim
4715   {
4716     \l_@@_x_initial_dim
4717     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4718     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4719   }
4720 \dim_set:Nn \l_tmpb_dim
4721 {
4722   \l_@@_y_initial_dim
4723   + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4724   * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4725 }
4726 \dim_set:Nn \l_@@_tmpc_dim
4727 {
4728   \l_@@_x_final_dim
4729   - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4730   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4731 }
4732 \dim_set:Nn \l_@@_tmpd_dim
4733 {
4734   \l_@@_y_final_dim
4735   - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4736   * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4737 }
4738 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4739 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4740 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4741 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4742 }

```

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

```

4743 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4744 {
4745   \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.

```

4746   \dim_zero_new:N \l_@@_l_dim
4747   \dim_set:Nn \l_@@_l_dim
4748   {
4749     \fp_to_dim:n
4750     {
4751       sqrt
4752       (
4753         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4754         +
4755         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4756       )
4757     }
4758   }

```

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.

```

4759   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4760   {
4761     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4762     \@@_draw_standard_dotted_line_i:
4763   }
4764 \group_end:
4765 \bool_lazy_all:nF
4766 {
4767   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4768   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4769   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4770     }
4771     \l_@@_labels_standard_dotted_line:
4772 }
4773 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4774 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4775 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

4776     \int_set:Nn \l_1tmpa_int
4777     {
4778         \dim_ratio:nn
4779         {
4780             \l_@@_l_dim
4781             - \l_@@_xdots_shorten_start_dim
4782             - \l_@@_xdots_shorten_end_dim
4783         }
4784         \l_@@_xdots_inter_dim
4785     }

```

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.

```

4786     \dim_set:Nn \l_1tmpa_dim
4787     {
4788         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4789         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4790     }
4791     \dim_set:Nn \l_1tmpb_dim
4792     {
4793         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4794         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4795     }

```

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.

```

4796     \dim_gadd:Nn \l_@@_x_initial_dim
4797     {
4798         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4799         \dim_ratio:nn
4800         {
4801             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4802             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4803         }
4804         { 2 \l_@@_l_dim }
4805     }
4806     \dim_gadd:Nn \l_@@_y_initial_dim
4807     {
4808         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4809         \dim_ratio:nn
4810         {
4811             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4812             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4813         }
4814         { 2 \l_@@_l_dim }
4815     }
4816     \pgf@relevantforpicturesizefalse
4817     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4818     {
4819         \pgfpathcircle
4820         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4821         { \l_@@_xdots_radius_dim }
4822         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4823         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4824     }

```

```

4825   \pgfusepathqfill
4826 }

4827 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4828 {
4829   \pgfscope
4830   \pgftransformshift
4831   {
4832     \pgfpointlineattime { 0.5 }
4833     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4834     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4835   }
4836   \fp_set:Nn \l_tmpa_fp
4837   {
4838     atand
4839     (
4840       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4841       \l_@@_x_final_dim - \l_@@_x_initial_dim
4842     )
4843   }
4844   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4845   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4846   \tl_if_empty:NF \l_@@_xdots_middle_tl
4847   {
4848     \begin { pgfscope }
4849     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4850     \pgfnode
4851     { rectangle }
4852     { center }
4853     {
4854       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4855       {
4856         \c_math_toggle_token
4857         \scriptstyle \l_@@_xdots_middle_tl
4858         \c_math_toggle_token
4859       }
4860     }
4861     { }
4862     {
4863       \pgfsetfillcolor { white }
4864       \pgfusepath { fill }
4865     }
4866     \end { pgfscope }
4867   }
4868   \tl_if_empty:NF \l_@@_xdots_up_tl
4869   {
4870     \pgfnode
4871     { rectangle }
4872     { south }
4873     {
4874       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4875       {
4876         \c_math_toggle_token
4877         \scriptstyle \l_@@_xdots_up_tl
4878         \c_math_toggle_token
4879       }
4880     }
4881     { }
4882     { \pgfusepath { } }
4883   }
4884   \tl_if_empty:NF \l_@@_xdots_down_tl
4885   {
4886     \pgfnode

```

```

4887     { rectangle }
4888     { north }
4889     {
4890     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4891     {
4892     \c_math_toggle_token
4893     \scriptstyle \l_@@_xdots_down_tl
4894     \c_math_toggle_token
4895     }
4896     }
4897     { }
4898     { \pgfusepath { } }
4899   }
4900 \endpgfscope
4901 }

```

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.

```

4902 \hook_gput_code:nnn { begindocument } { . }
4903 {
4904   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4905   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4906   \cs_new_protected:Npn \@@_Ldots
4907     { \@@_collect_options:n { \@@_Ldots_i } }
4908   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4909     {
4910     \int_if_zero:nTF \c@jCol
4911     { \@@_error:nn { in~first~col } \Ldots }
4912     {
4913     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4914     { \@@_error:nn { in~last~col } \Ldots }
4915     {
4916     \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4917     { #1 , down = #2 , up = #3 , middle = #4 }
4918     }
4919     }
4920   \bool_if:NF \l_@@_nullify_dots_bool
4921     { \phantom { \ensuremath { \@@_old_ldots } } }
4922   \bool_gset_true:N \g_@@_empty_cell_bool
4923 }

4924 \cs_new_protected:Npn \@@_Cdots
4925   { \@@_collect_options:n { \@@_Cdots_i } }
4926 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4927   {
4928   \int_if_zero:nTF \c@jCol
4929   { \@@_error:nn { in~first~col } \Cdots }
4930   {
4931   \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```

```

4932         { \@@_error:nn { in~last~col } \Cdots }
4933         {
4934             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4935             { #1 , down = #2 , up = #3 , middle = #4 }
4936         }
4937     }
4938     \bool_if:NF \l_@@_nullify_dots_bool
4939     { \phantom { \ensuremath { \@@_old_cdots } } }
4940     \bool_gset_true:N \g_@@_empty_cell_bool
4941 }

4942 \cs_new_protected:Npn \@@_Vdots
4943 { \@@_collect_options:n { \@@_Vdots_i } }
4944 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4945 {
4946     \int_if_zero:nTF \c@iRow
4947     { \@@_error:nn { in~first~row } \Vdots }
4948     {
4949         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4950         { \@@_error:nn { in~last~row } \Vdots }
4951         {
4952             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4953             { #1 , down = #2 , up = #3 , middle = #4 }
4954         }
4955     }
4956     \bool_if:NF \l_@@_nullify_dots_bool
4957     { \phantom { \ensuremath { \@@_old_vdots } } }
4958     \bool_gset_true:N \g_@@_empty_cell_bool
4959 }

4960 \cs_new_protected:Npn \@@_Ddots
4961 { \@@_collect_options:n { \@@_Ddots_i } }
4962 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4963 {
4964     \int_case:nnF \c@iRow
4965     {
4966         0 { \@@_error:nn { in~first~row } \Ddots }
4967         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4968     }
4969     {
4970         \int_case:nnF \c@jCol
4971         {
4972             0 { \@@_error:nn { in~first~col } \Ddots }
4973             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4974         }
4975         {
4976             \keys_set_known:nn { nicematrix / Ddots } { #1 }
4977             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4978             { #1 , down = #2 , up = #3 , middle = #4 }
4979         }
4980     }
4981 }
4982 \bool_if:NF \l_@@_nullify_dots_bool
4983 { \phantom { \ensuremath { \@@_old_ddots } } }
4984 \bool_gset_true:N \g_@@_empty_cell_bool
4985 }

4986 \cs_new_protected:Npn \@@_Iddots
4987 { \@@_collect_options:n { \@@_Iddots_i } }
4988 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4989 {

```



```

4990     \int_case:nnF \c@iRow
4991     {
4992         0           { \@@_error:nn { in~first~row } \Iddots }
4993         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4994     }
4995     {
4996         \int_case:nnF \c@jCol
4997         {
4998             0           { \@@_error:nn { in~first~col } \Iddots }
4999             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5000         }
5001         {
5002             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5003             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5004             { #1 , down = #2 , up = #3 , middle = #4 }
5005         }
5006     }
5007     \bool_if:NF \l_@@_nullify_dots_bool
5008     { \phantom { \ensuremath { \@@_old_iddots } } }
5009     \bool_gset_true:N \g_@@_empty_cell_bool
5010 }
5011 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5012 \keys_define:nn { nicematrix / Ddots }
5013 {
5014     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5015     draw-first .default:n = true ,
5016     draw-first .value_forbidden:n = true
5017 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5018 \cs_new_protected:Npn \@@_Hspace:
5019 {
5020     \bool_gset_true:N \g_@@_empty_cell_bool
5021     \hspace
5022 }

```

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`.

```

5023 \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`.

```

5024 \cs_new:Npn \@@_Hdotsfor:
5025 {
5026     \bool_lazy_and:nnTF
5027     { \int_if_zero_p:n \c@jCol }
5028     { \int_if_zero_p:n \l_@@_first_col_int }
5029     {
5030         \bool_if:NTF \g_@@_after_col_zero_bool
5031         {
5032             \multicolumn { 1 } { c } { }
5033             \@@_Hdotsfor_i
5034         }
5035         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5036     }
5037 }

```

```

5038     \multicolumn { 1 } { c } { }
5039     \@@_Hdotsfor_i
5040   }
5041 }

```

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:`).

```

5042 \hook_gput_code:nnn { begindocument } { . }
5043 {
5044   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5045   \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.

```

5046   \cs_new_protected:Npn \@@_Hdotsfor_i
5047     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5048   \exp_args:NNno \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5049     {
5050       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5051       {
5052         \@@_Hdotsfor:nnnn
5053         { \int_use:N \c@iRow }
5054         { \int_use:N \c@jCol }
5055         { #2 }
5056         {
5057           #1 , #3 ,
5058           down = \exp_not:n { #4 } ,
5059           up = \exp_not:n { #5 } ,
5060           middle = \exp_not:n { #6 }
5061         }
5062       }
5063       \prg_replicate:nn { #2 - 1 }
5064       {
5065         &
5066         \multicolumn { 1 } { c } { }
5067         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5068       }
5069     }
5070 }

```

```

5071 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5072 {
5073   \bool_set_false:N \l_@@_initial_open_bool
5074   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5075   \int_set:Nn \l_@@_initial_i_int { #1 }
5076   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5077   \int_compare:nNnTF { #2 } = \c_one_int
5078     {
5079       \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5080       \bool_set_true:N \l_@@_initial_open_bool
5081     }
5082     {
5083       \cs_if_exist:cTF
5084         {
5085           pgf @ sh @ ns @ \@@_env:
5086           - \int_use:N \l_@@_initial_i_int
5087           - \int_eval:n { #2 - 1 }
5088         }
5089         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5090     }

```

```

5091         \int_set:Nn \l_@@_initial_j_int { #2 }
5092         \bool_set_true:N \l_@@_initial_open_bool
5093     }
5094 }
5095 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5096 {
5097     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5098     \bool_set_true:N \l_@@_final_open_bool
5099 }
5100 {
5101     \cs_if_exist:cTF
5102     {
5103         pgf @ sh @ ns @ \@@_env:
5104         - \int_use:N \l_@@_final_i_int
5105         - \int_eval:n { #2 + #3 }
5106     }
5107     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5108     {
5109         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5110         \bool_set_true:N \l_@@_final_open_bool
5111     }
5112 }
5113 \group_begin:
5114 \@@_open_shorten:
5115 \int_if_zero:nTF { #1 }
5116 { \color { nicematrix-first-row } }
5117 {
5118     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5119     { \color { nicematrix-last-row } }
5120 }
5121
5122 \keys_set:nn { nicematrix / xdots } { #4 }
5123 \@@_color:o \l_@@_xdots_color_tl
5124 \@@_actually_draw_Ldots:
5125 \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).

```

5126     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5127     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5128 }

5129 \hook_gput_code:nnn { begindocument } { . }
5130 {
5131     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5132     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5133     \cs_new_protected:Npn \@@_Vdotsfor:
5134     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5135     \exp_args:NNNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5136     {
5137         \bool_gset_true:N \g_@@_empty_cell_bool
5138         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5139         {
5140             \@@_Vdotsfor:nnnn
5141             { \int_use:N \c@iRow }
5142             { \int_use:N \c@jCol }
5143             { #2 }
5144             {
5145                 #1 , #3 ,
5146                 down = \exp_not:n { #4 } ,
5147                 up = \exp_not:n { #5 } ,

```

```

5148         middle = \exp_not:n { #6 }
5149     }
5150 }
5151 }
5152 }

```

```

5153 \cs_new_protected:Npn \l_@@_Vdotsfor:nmmn #1 #2 #3 #4
5154 {
5155     \bool_set_false:N \l_@@_initial_open_bool
5156     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5157     \int_set:Nn \l_@@_initial_j_int { #2 }
5158     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5159     \int_compare:nNnTF { #1 } = \c_one_int
5160     {
5161         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5162         \bool_set_true:N \l_@@_initial_open_bool
5163     }
5164     {
5165         \cs_if_exist:cTF
5166         {
5167             pgf @ sh @ ns @ \l_@@_env:
5168             - \int_eval:n { #1 - 1 }
5169             - \int_use:N \l_@@_initial_j_int
5170         }
5171         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5172         {
5173             \int_set:Nn \l_@@_initial_i_int { #1 }
5174             \bool_set_true:N \l_@@_initial_open_bool
5175         }
5176     }
5177     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5178     {
5179         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5180         \bool_set_true:N \l_@@_final_open_bool
5181     }
5182     {
5183         \cs_if_exist:cTF
5184         {
5185             pgf @ sh @ ns @ \l_@@_env:
5186             - \int_eval:n { #1 + #3 }
5187             - \int_use:N \l_@@_final_j_int
5188         }
5189         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5190         {
5191             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5192             \bool_set_true:N \l_@@_final_open_bool
5193         }
5194     }
5195     \group_begin:
5196     \l_@@_open_shorten:
5197     \int_if_zero:nTF { #2 }
5198     { \color { nicematrix-first-col } }
5199     {
5200         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5201         { \color { nicematrix-last-col } }
5202     }
5203     \keys_set:nn { nicematrix / xdots } { #4 }
5204     \l_@@_color:o \l_@@_xdots_color_tl
5205     \l_@@_actually_draw_Vdots:
5206     \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).

```
5207 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5208   { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5209 }
```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```
5210 \NewDocumentCommand \@@_rotate: { 0 { } }
5211 {
5212   \peek_remove_spaces:n
5213   {
5214     \bool_gset_true:N \g_@@_rotate_bool
5215     \keys_set:nn { nicematrix / rotate } { #1 }
5216   }
5217 }

5218 \keys_define:nn { nicematrix / rotate }
5219 {
5220   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5221   c .value_forbidden:n = true ,
5222   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5223 }
```

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

```
5224 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5225 {
5226   \tl_if_empty:nTF { #2 }
5227   { #1 }
5228   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5229 }
5230 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5231 { \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).

```
5232 \hook_gput_code:nnn { begindocument } { . }
5233 {
```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5234 \cs_set_nopar:Npn \l_@@_argspec_tl
5235   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5236 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5237 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5238   {
5239     \group_begin:
5240     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5241     \@@_color:o \l_@@_xdots_color_tl
5242     \use:e
5243     {
5244       \@@_line_i:nn
5245       { \@@_double_int_eval:n #2 - \q_stop }
5246       { \@@_double_int_eval:n #3 - \q_stop }
5247     }
5248     \group_end:
5249   }
5250 }

5251 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5252 {
5253   \bool_set_false:N \l_@@_initial_open_bool
5254   \bool_set_false:N \l_@@_final_open_bool
5255   \bool_lazy_or:nnTF
5256     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5257     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5258     { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5259   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5260 }

5261 \hook_gput_code:nnn { begindocument } { . }
5262 {
5263   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5264   {

```

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:.`

```

5265     \c_@@_pgfortikzpicture_tl
5266     \@@_draw_line_iii:nn { #1 } { #2 }
5267     \c_@@_endpgfortikzpicture_tl
5268   }
5269 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5270 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5271 {
5272   \pgfrememberpicturepositiononpagetrue
5273   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5274   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5275   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5276   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5277   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5278   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5279   \@@_draw_line:
5280 }

```

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`

```
5281 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5282   { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```
5283 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5284 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5285   {
5286     \tl_gput_right:Ne \g_@@_row_style_tl
5287     {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5288     \exp_not:N
5289     \@@_if_row_less_than:nn
5290     { \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`).

```
5291     { \exp_not:n { #1 } \scan_stop: }
5292   }
5293 }
```

```
5294 \keys_define:nn { nicematrix / RowStyle }
5295   {
5296     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5297     cell-space-top-limit .value_required:n = true ,
5298     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5299     cell-space-bottom-limit .value_required:n = true ,
5300     cell-space-limits .meta:n =
5301     {
5302       cell-space-top-limit = #1 ,
5303       cell-space-bottom-limit = #1 ,
5304     } ,
5305     color .tl_set:N = \l_@@_color_tl ,
5306     color .value_required:n = true ,
5307     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5308     bold .default:n = true ,
5309     nb-rows .code:n =
5310     \str_if_eq:eeTF { #1 } { * }
5311     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5312     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5313     nb-rows .value_required:n = true ,
5314     rowcolor .tl_set:N = \l_tmpa_tl ,
5315     rowcolor .value_required:n = true ,
5316     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5317   }
```

```

5318 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5319 {
5320   \group_begin:
5321   \tl_clear:N \l_tmpa_tl
5322   \tl_clear:N \l_@@_color_tl
5323   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5324   \dim_zero:N \l_tmpa_dim
5325   \dim_zero:N \l_tmpb_dim
5326   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5327   \tl_if_empty:NF \l_tmpa_tl
5328   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5329     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5330     {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5331         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5332         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5333         { \int_use:N \c@iRow - * }
5334     }

```

Then, the other rows (if there is several rows).

```

5335     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5336     {
5337       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5338       {
5339         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5340         {
5341           \int_eval:n { \c@iRow + 1 }
5342           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5343         }
5344       }
5345     }
5346   }
5347   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5348     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5349     {
5350       \@@_put_in_row_style:e
5351       {
5352         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5353         {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5354             \dim_set:Nn \l_@@_cell_space_top_limit_dim
5355             { \dim_use:N \l_tmpa_dim }
5356         }
5357     }
5358 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5359     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5360     {
5361       \@@_put_in_row_style:e
5362       {
5363         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5364         {
5365           \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5366           { \dim_use:N \l_tmpb_dim }
5367         }
5368       }
5369     }

```


`\l_@@_color_tl` is the value of the key color of `\RowStyle`.

```

5370   \tl_if_empty:NF \l_@@_color_tl
5371   {
5372     \@@_put_in_row_style:e
5373     {
5374       \mode_leave_vertical:
5375       \@@_color:n { \l_@@_color_tl }
5376     }
5377   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5378   \bool_if:NT \l_@@_bold_row_style_bool
5379   {
5380     \@@_put_in_row_style:n
5381     {
5382       \exp_not:n
5383       {
5384         \if_mode_math:
5385         \c_math_toggle_token
5386         \bfseries \boldmath
5387         \c_math_toggle_token
5388         \else:
5389         \bfseries \boldmath
5390         \fi:
5391       }
5392     }
5393   }
5394   \group_end:
5395   \g_@@_row_style_tl
5396   \ignorespaces
5397 }

```

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).

```

5398 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5399 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5400 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5401 {

```

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.

```
5402 \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`.

```
5403 \str_if_in:nnF { #1 } { !! }
5404 {
5405 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5406 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5407 }
5408 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5409 {
5410 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5411 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5412 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5413 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5414 }
```

The following command must be used within a `\pgfpicture`.

```
5415 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5416 {
5417 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5418 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5419 \group_begin:
5420 \pgfsetcornersarced
5421 {
5422 \pgfpoint
5423 { \l_@@_tab_rounded_corners_dim }
5424 { \l_@@_tab_rounded_corners_dim }
5425 }
```

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.

```
5426 \bool_if:NTF \l_@@_hvlines_bool
5427 {
5428 \pgfpathrectanglecorners
5429 {
5430 \pgfpointadd
5431 { \@@_qpoint:n { row-1 } }
5432 { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5433 }
5434 {
5435 \pgfpointadd
5436 {
5437 \@@_qpoint:n
5438 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5439 }
5440 { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5441 }
5442 }
5443 {
```

```

5444     \pgfpathrectanglecorners
5445     { \@@_qpoint:n { row-1 } }
5446     {
5447     \pgfpointadd
5448     {
5449     \@@_qpoint:n
5450     { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5451     }
5452     { \pgfpoint \c_zero_dim \arrayrulewidth }
5453     }
5454     }
5455     \pgfusepath { clip }
5456     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5457     }
5458 }

```

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`).

```

5459 \cs_new_protected:Npn \@@_actually_color:
5460 {
5461 \pgfpicture
5462 \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.

```

5463 \@@_clip_with_rounded_corners:
5464 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5465 {
5466 \int_compare:nNnTF { ##1 } = \c_one_int
5467 {
5468 \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5469 \use:c { g_@@_color _ 1 _tl }
5470 \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5471 }
5472 {
5473 \begin { pgfscope }
5474 \@@_color_opacity ##2
5475 \use:c { g_@@_color _ ##1 _tl }
5476 \tl_gclear:c { g_@@_color _ ##1 _tl }
5477 \pgfusepath { fill }
5478 \end { pgfscope }
5479 }
5480 }
5481 \endpgfpicture
5482 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5483 \cs_new_protected:Npn \@@_color_opacity
5484 {
5485 \peek_meaning:NTF [
5486 { \@@_color_opacity:w }
5487 { \@@_color_opacity:w [ ] }
5488 }

```

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.

```

5489 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5490 {
5491 \tl_clear:N \l_tmpa_tl
5492 \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.

```

5493     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5494     \tl_if_empty:NTF \l_tmpb_tl
5495         { \@declaredcolor }
5496         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5497     }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5498 \keys_define:nn { nicematrix / color-opacity }
5499 {
5500     opacity .tl_set:N          = \l_tmpa_tl ,
5501     opacity .value_required:n = true
5502 }

5503 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5504 {
5505     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5506     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5507     \@@_cartesian_path:
5508 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5509 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5510 {
5511     \tl_if_blank:nF { #2 }
5512     {
5513         \@@_add_to_colors_seq:en
5514         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5515         { \@@_cartesian_color:nn { #3 } { - } }
5516     }
5517 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5518 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5519 {
5520     \tl_if_blank:nF { #2 }
5521     {
5522         \@@_add_to_colors_seq:en
5523         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5524         { \@@_cartesian_color:nn { - } { #3 } }
5525     }
5526 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5527 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5528 {
5529     \tl_if_blank:nF { #2 }
5530     {
5531         \@@_add_to_colors_seq:en
5532         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5533         { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5534     }
5535 }

```

The last argument is the radius of the corners of the rectangle.

```

5536 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5537 {
5538     \tl_if_blank:nF { #2 }

```

```

5539     {
5540     \@@_add_to_colors_seq:en
5541     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5542     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5543     }
5544 }

```

The last argument is the radius of the corners of the rectangle.

```

5545 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5546 {
5547   \@@_cut_on_hyphen:w #1 \q_stop
5548   \tl_clear_new:N \l_@@_tmpc_tl
5549   \tl_clear_new:N \l_@@_tmpd_tl
5550   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5551   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5552   \@@_cut_on_hyphen:w #2 \q_stop
5553   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5554   \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`.

```

5555   \@@_cartesian_path:n { #3 }
5556 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5557 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5558 {
5559   \clist_map_inline:nn { #3 }
5560   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5561 }

5562 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5563 {
5564   \int_step_inline:nn \c@iRow
5565   {
5566     \int_step_inline:nn \c@jCol
5567     {
5568       \int_if_even:nTF { #####1 + ##1 }
5569       { \@@_cellcolor [ #1 ] { #2 } }
5570       { \@@_cellcolor [ #1 ] { #3 } }
5571       { ##1 - #####1 }
5572     }
5573   }
5574 }

```

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”.

```

5575 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5576 {
5577   \@@_rectanglecolor [ #1 ] { #2 }
5578   { 1 - 1 }
5579   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5580 }

5581 \keys_define:nn { nicematrix / rowcolors }
5582 {
5583   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5584   respect-blocks .default:n = true ,
5585   cols .tl_set:N = \l_@@_cols_tl ,

```

```

5586     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5587     restart .default:n = true ,
5588     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5589 }

```

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`.

```

5590 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5591 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5592     \group_begin:
5593     \seq_clear_new:N \l_@@_colors_seq
5594     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5595     \tl_clear_new:N \l_@@_cols_tl
5596     \cs_set_nopar:Npn \l_@@_cols_tl { - }
5597     \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).

```

5598     \int_zero_new:N \l_@@_color_int
5599     \int_set_eq:NN \l_@@_color_int \c_one_int
5600     \bool_if:NT \l_@@_respect_blocks_bool
5601     {

```

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`).

```

5602         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5603         \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5604         { \@@_not_in_exterior_p:nnnn ##1 }
5605     }
5606     \pgfpicture
5607     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5608     \clist_map_inline:nn { #2 }
5609     {
5610         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5611         \tl_if_in:NnTF \l_tmpa_tl { - }
5612         { \@@_cut_on_hyphen:w ##1 \q_stop }
5613         { \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.

```

5614         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5615         \int_set:Nn \l_@@_color_int
5616         { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5617         \int_zero_new:N \l_@@_tmpc_int
5618         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5619         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5620         {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5621             \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).

```

5622         \bool_if:NT \l_@@_respect_blocks_bool
5623         {
5624             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5625             { \@@_intersect_our_row_p:nnnn #####1 }
5626             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5627     }
5628     \tl_set:No \l_@@_rows_tl
5629     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
5630     \tl_clear_new:N \l_@@_color_tl
5631     \tl_set:Ne \l_@@_color_tl
5632     {
5633       \@@_color_index:n
5634       {
5635         \int_mod:nn
5636         { \l_@@_color_int - 1 }
5637         { \seq_count:N \l_@@_colors_seq }
5638         + 1
5639       }
5640     }
5641     \tl_if_empty:NF \l_@@_color_tl
5642     {
5643       \@@_add_to_colors_seq:ee
5644       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5645       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5646     }
5647     \int_incr:N \l_@@_color_int
5648     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5649   }
5650 }
5651 \endpgfpicture
5652 \group_end:
5653 }

```

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.

```

5654 \cs_new:Npn \@@_color_index:n #1
5655 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5656   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5657   { \@@_color_index:n { #1 - 1 } }
5658   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5659 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5660 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5661 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5662 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5663 {
5664   \int_compare:nNnT { #3 } > \l_tmpb_int
5665   { \int_set:Nn \l_tmpb_int { #3 } }
5666 }

```

```

5667 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5668 {
5669   \int_if_zero:nTF { #4 }
5670   \prg_return_false:
5671   {
5672     \int_compare:nNnTF { #2 } > \c@jCol

```

```

5673     \prg_return_false:
5674     \prg_return_true:
5675   }
5676 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5677 \prg_new_conditional:Nnn \@@_intersect_our_row:nmmnn p
5678 {
5679   \int_compare:nNnTF { #1 } > \l_tmpa_int
5680     \prg_return_false:
5681     {
5682       \int_compare:nNnTF \l_tmpa_int > { #3 }
5683       \prg_return_false:
5684       \prg_return_true:
5685     }
5686 }

```

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).

```

5687 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5688 {
5689   \dim_compare:nNnTF { #1 } = \c_zero_dim
5690     {
5691       \bool_if:NTF
5692         \l_@@_nocolor_used_bool
5693         \@@_cartesian_path_normal_ii:
5694       {
5695         \clist_if_empty:NTF \l_@@_corners_cells_clist
5696           { \@@_cartesian_path_normal_i:n { #1 } }
5697         \@@_cartesian_path_normal_ii:
5698       }
5699     }
5700   { \@@_cartesian_path_normal_i:n { #1 } }
5701 }

```

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.

```

5702 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5703 {
5704   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5705   \clist_map_inline:Nn \l_@@_cols_tl
5706   {
5707     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5708     \tl_if_in:NnTF \l_tmpa_tl { - }
5709     { \@@_cut_on_hyphen:w ##1 \q_stop }
5710     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5711     \tl_if_empty:NTF \l_tmpa_tl
5712     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5713     {
5714       \str_if_eq:eeT \l_tmpa_tl { * }
5715       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5716     }
5717     \tl_if_empty:NTF \l_tmpb_tl
5718     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5719     {

```



```

5720     \str_if_eq:eeT \l_tmpb_tl { * }
5721     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5722   }
5723   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5724     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
\l_@@_tmpc_tl will contain the number of column.

```

```

5725     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5726     \@@_qpoint:n { col - \l_tmpa_tl }
5727     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5728       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5729       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5730     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5731     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5732   \clist_map_inline:Nn \l_@@_rows_tl
5733   {
5734     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5735     \tl_if_in:NnTF \l_tmpa_tl { - }
5736       { \@@_cut_on_hyphen:w #####1 \q_stop }
5737       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5738     \tl_if_empty:NnTF \l_tmpa_tl
5739       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5740     {
5741       \str_if_eq:eeT \l_tmpa_tl { * }
5742       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5743     }
5744     \tl_if_empty:NnTF \l_tmpb_tl
5745       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5746     {
5747       \str_if_eq:eeT \l_tmpb_tl { * }
5748       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5749     }
5750     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5751     { \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.

```

5752     \cs_if_exist:cF
5753     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5754     {
5755       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5756       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5757       \@@_qpoint:n { row - \l_tmpa_tl }
5758       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5759       \pgfpathrectanglecorners
5760         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5761         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5762     }
5763   }
5764 }
5765 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5766 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5767 {
5768   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5769   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5770   \clist_map_inline:Nn \l_@@_cols_tl
5771   {
5772     \@@_qpoint:n { col - ##1 }
5773     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }

```

```

5774     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5775     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5776     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5777     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5778     \clist_map_inline:Nn \l_@@_rows_tl
5779     {
5780         \@@_if_in_corner:nF { #####1 - ##1 }
5781         {
5782             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5783             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5784             \@@_qpoint:n { row - #####1 }
5785             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5786             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5787             {
5788                 \pgfpathrectanglecorners
5789                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5790                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5791             }
5792         }
5793     }
5794 }
5795 }

```

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`).

```

5796 \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

```

5797 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5798 {
5799     \bool_set_true:N \l_@@_nocolor_used_bool
5800     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5801     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5802     \clist_map_inline:Nn \l_@@_rows_tl
5803     {
5804         \clist_map_inline:Nn \l_@@_cols_tl
5805         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5806     }
5807 }

```

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.

```

5808 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5809 {
5810     \clist_set_eq:NN \l_tmpa_clist #1
5811     \clist_clear:N #1
5812     \clist_map_inline:Nn \l_tmpa_clist
5813     {
5814         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5815         \tl_if_in:NnTF \l_tmpa_tl { - }
5816         { \@@_cut_on_hyphen:w ##1 \q_stop }
5817         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5818         \bool_lazy_or:nnT
5819         { \str_if_eq_p:ee \l_tmpa_tl { * } }
5820         { \tl_if_blank_p:o \l_tmpa_tl }

```

```

5821     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5822 \bool_lazy_or:nnT
5823   { \str_if_eq_p:ee \l_tmpb_tl { * } }
5824   { \tl_if_blank_p:o \l_tmpb_tl }
5825   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5826 \int_compare:nNnT \l_tmpb_tl > #2
5827   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5828 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5829   { \clist_put_right:Nn #1 { #####1 } }
5830 }
5831 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5832 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5833 {
5834   \@@_test_color_inside:
5835   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5836   {

```

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`).

```

5837     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5838     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5839   }
5840   \ignorespaces
5841 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5842 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5843 {
5844   \@@_test_color_inside:
5845   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5846   {
5847     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5848     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5849     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5850   }
5851   \ignorespaces
5852 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5853 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5854 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5855 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5856 {
5857   \@@_test_color_inside:
5858   \peek_remove_spaces:n
5859   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5860 }

5861 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5862 {

```

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`.

```

5863     \seq_gclear:N \g_tmpa_seq
5864     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5865       { \@@_rowlistcolors_tabular_i:nmmn ##1 }
5866     \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`.

```

5867     \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5868     {
5869       { \int_use:N \c@iRow }
5870       { \exp_not:n { #1 } }
5871       { \exp_not:n { #2 } }
5872       { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5873     }
5874   }

```

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`).

```

5875     \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nmmn #1 #2 #3 #4
5876     {
5877       \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).

```

5878       { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5879     {
5880       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5881       {
5882         \@@_rowlistcolors
5883         [ \exp_not:n { #2 } ]
5884         { #1 - \int_eval:n { \c@iRow - 1 } }
5885         { \exp_not:n { #3 } }
5886         [ \exp_not:n { #4 } ]
5887       }
5888     }
5889   }

```

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.

```

5890     \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5891     {
5892       \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5893       { \@@_rowlistcolors_tabular_ii:nmmn ##1 }
5894       \seq_gclear:N \g_@@_rowlistcolors_seq
5895     }

```

```

5896 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nmmm #1 #2 #3 #4
5897 {
5898   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5899   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5900 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is written in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5901 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5902 {

```

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).

```

5903   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5904   {

```

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).

```

5905     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5906     {
5907       \exp_not:N \columncolor [ #1 ]
5908       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5909     }
5910   }
5911 }

```

```

5912 \hook_gput_code:nnn { begindocument } { . }
5913 {
5914   \IfPackageLoadedTF { colortbl }
5915   {
5916     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5917     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5918     \cs_new_protected:Npn \@@_revert_colortbl:
5919     {
5920       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5921       {
5922         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5923         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5924       }
5925     }
5926   }
5927   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5928 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of array) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
5929 \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`.

```
5930 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5931 {
5932   \int_if_zero:nTF \l_@@_first_col_int
5933     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5934     {
5935       \int_if_zero:nTF \c@jCol
5936         {
5937           \int_compare:nNnF \c@iRow = { -1 }
5938             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5939         }
5940         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5941     }
5942 }
```

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.

```
5943 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5944 {
5945   \int_if_zero:nF \c@iRow
5946   {
5947     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5948     {
5949       \int_compare:nNnT \c@jCol > \c_zero_int
5950       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5951     }
5952   }
5953 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5954 \keys_define:nn { nicematrix / Rules }
5955 {
5956   position .int_set:N = \l_@@_position_int ,
5957   position .value_required:n = true ,
5958   start .int_set:N = \l_@@_start_int ,
5959   end .code:n =
5960     \bool_lazy_or:nnTF
5961       { \tl_if_empty_p:n { #1 } }
5962       { \str_if_eq_p:ee { #1 } { last } }
5963       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5964       { \int_set:Nn \l_@@_end_int { #1 } }
5965 }
```

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.

```

5966 \keys_define:nn { nicematrix / RulesBis }
5967   {
5968     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5969     multiplicity .initial:n = 1 ,
5970     dotted .bool_set:N = \l_@@_dotted_bool ,
5971     dotted .initial:n = false ,
5972     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]`.

```

5973   color .code:n =
5974     \@@_set_CT@arc@:n { #1 }
5975     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5976   color .value_required:n = true ,
5977   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5978   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.

```

5979   tikz .code:n =
5980     \IfPackageLoadedTF { tikz }
5981       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5982       { \@@_error:n { tikz-without-tikz } } ,
5983   tikz .value_required:n = true ,
5984   total-width .dim_set:N = \l_@@_rule_width_dim ,
5985   total-width .value_required:n = true ,
5986   width .meta:n = { total-width = #1 } ,
5987   unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5988   }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5989 \cs_new_protected:Npn \@@_vline:n #1
5990   {

```

The group is for the options.

```

5991   \group_begin:
5992   \int_set_eq:NN \l_@@_end_int \c@iRow
5993   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5994   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5995     \@@_vline_i:
5996   \group_end:
5997   }

5998 \cs_new_protected:Npn \@@_vline_i:
5999   {

```

`\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`.

```

6000   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6001   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6002     \l_tmpa_tl
6003   {

```

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.

```

6004     \bool_gset_true:N \g_tmpa_bool
6005     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6006       { \@@_test_vline_in_block:nnnnn ##1 }
6007     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6008       { \@@_test_vline_in_block:nnnnn ##1 }
6009     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6010       { \@@_test_vline_in_stroken_block:nnnn ##1 }
6011     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6012     \bool_if:NTF \g_tmpa_bool
6013       {
6014         \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.

```

6015         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6016       }
6017     {
6018       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6019       {
6020         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6021         \@@_vline_ii:
6022         \int_zero:N \l_@@_local_start_int
6023       }
6024     }
6025   }
6026   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6027   {
6028     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6029     \@@_vline_ii:
6030   }
6031 }

6032 \cs_new_protected:Npn \@@_test_in_corner_v:
6033 {
6034   \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6035   {
6036     \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6037     { \bool_set_false:N \g_tmpa_bool }
6038   }
6039   {
6040     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6041     {
6042       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6043       { \bool_set_false:N \g_tmpa_bool }
6044       {
6045         \@@_if_in_corner:nT
6046           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6047           { \bool_set_false:N \g_tmpa_bool }
6048       }
6049     }
6050   }
6051 }

6052 \cs_new_protected:Npn \@@_vline_ii:
6053 {
6054   \tl_clear:N \l_@@_tikz_rule_tl
6055   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```



```

6056 \bool_if:NTF \l_@@_dotted_bool
6057   \@@_vline_iv:
6058   {
6059     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6060     \@@_vline_iii:
6061     \@@_vline_v:
6062   }
6063 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6064 \cs_new_protected:Npn \@@_vline_iii:
6065 {
6066   \pgfpicture
6067   \pgfrememberpicturepositiononpagetrue
6068   \pgf@relevantforpicturesizefalse
6069   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6070   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6071   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6072   \dim_set:Nn \l_tmpb_dim
6073   {
6074     \pgf@x
6075     - 0.5 \l_@@_rule_width_dim
6076     +
6077     ( \arrayrulewidth * \l_@@_multiplicity_int
6078       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6079   }
6080   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6081   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6082   \bool_lazy_all:nT
6083   {
6084     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6085     { \cs_if_exist_p:N \CT@drsc@ }
6086     { ! \tl_if_blank_p:o \CT@drsc@ }
6087   }
6088   {
6089     \group_begin:
6090     \CT@drsc@
6091     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6092     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6093     \dim_set:Nn \l_@@_tmpd_dim
6094     {
6095       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6096       * ( \l_@@_multiplicity_int - 1 )
6097     }
6098     \pgfpathrectanglecorners
6099     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6100     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6101     \pgfusepath { fill }
6102     \group_end:
6103   }
6104   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6105   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6106   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6107   {
6108     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6109     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6110     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6111     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6112   }
6113   \CT@arc@
6114   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6115   \pgfsetrectcap
6116   \pgfusepathqstroke

```

```

6117 \endpgfpicture
6118 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6119 \cs_new_protected:Npn \@@_vline_iv:
6120 {
6121   \pgfpicture
6122   \pgfrememberpicturepositiononpagetrue
6123   \pgf@relevantforpicturesizefalse
6124   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6125   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6126   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6127   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6128   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6129   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6130   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6131   \CT@arc@
6132   \@@_draw_line:
6133   \endpgfpicture
6134 }

```

The following code is for the case when the user uses the key `tikz`.

```

6135 \cs_new_protected:Npn \@@_vline_v:
6136 {
6137   \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).

```

6138   \CT@arc@
6139   \tl_if_empty:NF \l_@@_rule_color_tl
6140   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6141   \pgfrememberpicturepositiononpagetrue
6142   \pgf@relevantforpicturesizefalse
6143   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6144   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6145   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6146   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6147   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6148   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6149   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6150   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6151   ( \l_tmpb_dim , \l_tmpa_dim ) --
6152   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6153   \end {tikzpicture }
6154 }

```

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).

```

6155 \cs_new_protected:Npn \@@_draw_vlines:
6156 {
6157   \int_step_inline:nnn
6158   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6159   {
6160     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6161     \c@jCol
6162     { \int_eval:n { \c@jCol + 1 } }
6163   }
6164   {
6165     \str_if_eq:eeF \l_@@_vlines_clist { all }
6166     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6167     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6168     }
6169 }

```

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}`.

```

6170 \cs_new_protected:Npn \@@_hline:n #1
6171 {

```

The group is for the options.

```

6172     \group_begin:
6173     \int_zero_new:N \l_@@_end_int
6174     \int_set_eq:NN \l_@@_end_int \c@jCol
6175     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6176     \@@_hline_i:
6177     \group_end:
6178 }

```

```

6179 \cs_new_protected:Npn \@@_hline_i:

```

```

6180 {
6181     \int_zero_new:N \l_@@_local_start_int
6182     \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`.

```

6183     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6184     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6185         \l_tmpb_tl
6186     {

```

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.

```

6187         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6188         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6189             { \@@_test_hline_in_block:nnnnn ##1 }
6190         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6191             { \@@_test_hline_in_block:nnnnn ##1 }
6192         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6193             { \@@_test_hline_in_stroken_block:nnnn ##1 }
6194         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6195         \bool_if:NTF \g_tmpa_bool
6196             {
6197             \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.

```

6198             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6199         }
6200     {
6201         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6202         {
6203             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6204             \@@_hline_ii:
6205             \int_zero:N \l_@@_local_start_int
6206         }
6207     }
6208 }
6209 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6210     {
6211     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6212     \@@_hline_ii:
6213     }
6214 }

6215 \cs_new_protected:Npn \@@_test_in_corner_h:
6216 {
6217     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6218     {
6219     \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6220     { \bool_set_false:N \g_tmpa_bool }
6221     }
6222     {
6223     \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6224     {
6225     \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6226     { \bool_set_false:N \g_tmpa_bool }
6227     {
6228     \@@_if_in_corner:nT
6229     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6230     { \bool_set_false:N \g_tmpa_bool }
6231     }
6232     }
6233     }
6234 }

6235 \cs_new_protected:Npn \@@_hline_ii:
6236 {
6237     \tl_clear:N \l_@@_tikz_rule_tl
6238     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6239     \bool_if:NTF \l_@@_dotted_bool
6240     \@@_hline_iv:
6241     {
6242     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6243     \@@_hline_iii:
6244     \@@_hline_v:
6245     }
6246 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6247 \cs_new_protected:Npn \@@_hline_iii:
6248 {
6249     \pgfpicture
6250     \pgfrememberpicturepositiononpagetrue
6251     \pgf@relevantforpicturesizefalse
6252     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6253     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6254     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6255     \dim_set:Nn \l_tmpb_dim
6256     {
6257     \pgf@y
6258     - 0.5 \l_@@_rule_width_dim
6259     +
6260     ( \arrayrulewidth * \l_@@_multiplicity_int
6261     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6262     }
6263     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6264     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6265     \bool_lazy_all:nT
6266     {

```

```

6267     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6268     { \cs_if_exist_p:N \CT@drsc@ }
6269     { ! \tl_if_blank_p:o \CT@drsc@ }
6270   }
6271   {
6272     \group_begin:
6273     \CT@drsc@
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_tmpa_dim \l_tmpb_dim }
6281     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6282     \pgfusepathqfill
6283     \group_end:
6284   }
6285   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6286   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_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_tmpa_dim \l_tmpb_dim }
6292     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_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). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6300 \cs_new_protected:Npn \@@_hline_iv:
6301   {
6302     \pgfpicture
6303     \pgfrememberpicturepositiononpagetrue
6304     \pgf@relevantforpicturesizefalse
6305     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6306     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6307     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6308     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6309     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

6310 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6311 {
6312   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6313   \bool_if:NF \g_@@_delims_bool
6314   { \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.

```

6315   \tl_if_eq:NnF \g_@@_left_delim_tl (
6316     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6317   )
6318   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6319   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6320   \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6321   {
6322     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6323     \bool_if:NF \g_@@_delims_bool
6324     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6325     \tl_if_eq:NnF \g_@@_right_delim_tl )
6326     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6327   }
6328   \CT@arc@
6329   \@@_draw_line:
6330   \endpgfpicture
6331 }

```

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`).

```

6332 \cs_new_protected:Npn \@@_hline_v:
6333 {
6334   \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).

```

6335   \CT@arc@
6336   \tl_if_empty:NF \l_@@_rule_color_tl
6337   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6338   \pgfrememberpicturepositiononpagetrue
6339   \pgf@relevantforpicturesizefalse
6340   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6341   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6342   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6343   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6344   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6345   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6346   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6347   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6348   ( \l_tmpa_dim , \l_tmpb_dim ) --
6349   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6350   \end { tikzpicture }
6351 }

```

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).

```

6352 \cs_new_protected:Npn \@@_draw_hlines:
6353 {
6354   \int_step_inline:nnn
6355   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6356   {
6357     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6358     \c@iRow
6359     { \int_eval:n { \c@iRow + 1 } }
6360   }
6361   {
6362     \str_if_eq:eeF \l_@@_hlines_clist { all }
6363     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6364     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6365   }
6366 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6367 \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.

```

6368 \cs_set:Npn \@@_Hline_i:n #1
6369 {
6370   \peek_remove_spaces:n
6371   {
6372     \peek_meaning:NTF \Hline
6373     { \@@_Hline_ii:nn { #1 + 1 } }
6374     { \@@_Hline_iii:n { #1 } }
6375   }
6376 }
6377 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6378 \cs_set:Npn \@@_Hline_iii:n #1
6379 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6380 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6381 {
6382   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6383   \skip_vertical:N \l_@@_rule_width_dim
6384   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6385   {
6386     \@@_hline:n
6387     {
6388       multiplicity = #1 ,
6389       position = \int_eval:n { \c@iRow + 1 } ,
6390       total-width = \dim_use:N \l_@@_rule_width_dim ,
6391       #2
6392     }
6393   }
6394   \egroup
6395 }

```

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`).

```

6396 \cs_new_protected:Npn \@@_custom_line:n #1
6397 {
6398   \str_clear_new:N \l_@@_command_str
6399   \str_clear_new:N \l_@@_ccommand_str
6400   \str_clear_new:N \l_@@_letter_str
6401   \tl_clear_new:N \l_@@_other_keys_tl
6402   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6403 \bool_lazy_all:nTF
6404 {
6405   { \str_if_empty_p:N \l_@@_letter_str }
6406   { \str_if_empty_p:N \l_@@_command_str }
6407   { \str_if_empty_p:N \l_@@_ccommand_str }
6408 }
6409 { \@@_error:n { No~letter~and~no~command } }
6410 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6411 }
6412 \keys_define:nm { nicematrix / custom-line }
6413 {
6414   letter .str_set:N = \l_@@_letter_str ,
6415   letter .value_required:n = true ,
6416   command .str_set:N = \l_@@_command_str ,
6417   command .value_required:n = true ,
6418   ccommand .str_set:N = \l_@@_cccommand_str ,
6419   ccommand .value_required:n = true ,
6420 }
6421 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6422 \cs_new_protected:Npn \@@_custom_line_i:n #1
6423 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6424 \bool_set_false:N \l_@@_tikz_rule_bool
6425 \bool_set_false:N \l_@@_dotted_rule_bool
6426 \bool_set_false:N \l_@@_color_bool
6427 \keys_set:nm { nicematrix / custom-line-bis } { #1 }
6428 \bool_if:NT \l_@@_tikz_rule_bool
6429 {
6430   \IfPackageLoadedF { tikz }
6431   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6432   \bool_if:NT \l_@@_color_bool
6433   { \@@_error:n { color-in-custom-line-with-tikz } }
6434 }
6435 \bool_if:NT \l_@@_dotted_rule_bool
6436 {
6437   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6438   { \@@_error:n { key~multiplicity~with~dotted } }
6439 }
6440 \str_if_empty:NF \l_@@_letter_str
6441 {
6442   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6443   { \@@_error:n { Several~letters } }
6444   {
6445     \tl_if_in:NoTF
6446     \c_@@_forbidden_letters_str
6447     \l_@@_letter_str
6448     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6449     {

```

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.

```

6450 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6451 { \@@_v_custom_line:n { #1 } }
6452 }
6453 }
6454 }
6455 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6456 \str_if_empty:NF \l_@@_cccommand_str { \@@_c_custom_line:n { #1 } }
6457 }

```



```

6458 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|() []!@<> }
6459 \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.

```

6460 \keys_define:nn { nicematrix / custom-line-bis }
6461 {
6462   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6463   multiplicity .initial:n = 1 ,
6464   multiplicity .value_required:n = true ,
6465   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6466   color .value_required:n = true ,
6467   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6468   tikz .value_required:n = true ,
6469   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6470   dotted .value_forbidden:n = true ,
6471   total-width .code:n = { } ,
6472   total-width .value_required:n = true ,
6473   width .code:n = { } ,
6474   width .value_required:n = true ,
6475   sep-color .code:n = { } ,
6476   sep-color .value_required:n = true ,
6477   unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6478 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6479 \bool_new:N \l_@@_dotted_rule_bool
6480 \bool_new:N \l_@@_tikz_rule_bool
6481 \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`.

```

6482 \keys_define:nn { nicematrix / custom-line-width }
6483 {
6484   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6485   multiplicity .initial:n = 1 ,
6486   multiplicity .value_required:n = true ,
6487   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6488   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6489   \bool_set_true:N \l_@@_total_width_bool ,
6490   total-width .value_required:n = true ,
6491   width .meta:n = { total-width = #1 } ,
6492   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6493 }

```

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`).

```

6494 \cs_new_protected:Npn \@@_h_custom_line:n #1
6495 {

```

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`).

```

6496   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6497   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6498 }

```

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`).

```
6499 \cs_new_protected:Npn \@@_c_custom_line:n #1
6500 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6501   \exp_args:Nc \NewExpandableDocumentCommand
6502     { nicematrix - \l_@@_ccommand_str }
6503     { 0 { } m }
6504     {
6505       \noalign
6506       {
6507         \@@_compute_rule_width:n { #1 , ##1 }
6508         \skip_vertical:n { \l_@@_rule_width_dim }
6509         \clist_map_inline:nn
6510           { ##2 }
6511           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6512       }
6513     }
6514   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6515 }
```

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*.

```
6516 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6517 {
6518   \tl_if_in:nnTF { #2 } { - }
6519     { \@@_cut_on_hyphen:w #2 \q_stop }
6520     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6521   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6522     {
6523       \@@_hline:n
6524       {
6525         #1 ,
6526         start = \l_tmpa_tl ,
6527         end = \l_tmpb_tl ,
6528         position = \int_eval:n { \c@iRow + 1 } ,
6529         total-width = \dim_use:N \l_@@_rule_width_dim
6530       }
6531     }
6532 }

6533 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6534 {
6535   \bool_set_false:N \l_@@_tikz_rule_bool
6536   \bool_set_false:N \l_@@_total_width_bool
6537   \bool_set_false:N \l_@@_dotted_rule_bool
6538   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6539   \bool_if:NF \l_@@_total_width_bool
6540     {
6541       \bool_if:NTF \l_@@_dotted_rule_bool
6542         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6543         {
6544           \bool_if:NF \l_@@_tikz_rule_bool
6545             {
6546               \dim_set:Nn \l_@@_rule_width_dim
6547                 {
6548                   \arrayrulewidth * \l_@@_multiplicity_int
6549                   + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6550                 }
6551             }
6552         }
6553     }
6554 }
```

```

6555 \cs_new_protected:Npn \@@_v_custom_line:n #1
6556 {
6557   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6558   \tl_gput_right:Ne \g_@@_array_preamble_tl
6559     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6560   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6561     {
6562     \@@_vline:n
6563     {
6564       #1 ,
6565       position = \int_eval:n { \c@jCol + 1 } ,
6566       total-width = \dim_use:N \l_@@_rule_width_dim
6567     }
6568   }
6569   \@@_rec_preamble:n
6570 }

6571 \@@_custom_line:n
6572 { 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.

```

6573 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6574 {
6575   \int_compare:nNnT \l_tmpa_tl > { #1 }
6576   {
6577     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6578     {
6579       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6580       {
6581         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6582         { \bool_gset_false:N \g_tmpa_bool }
6583       }
6584     }
6585   }
6586 }

```

The same for vertical rules.

```

6587 \cs_new_protected:Npn \@@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6588 {
6589   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6590   {
6591     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6592     {
6593       \int_compare:nNnT \l_tmpb_tl > { #2 }
6594       {
6595         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6596         { \bool_gset_false:N \g_tmpa_bool }
6597       }
6598     }
6599   }
6600 }

6601 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6602 {
6603   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6604   {
6605     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6606     {

```

```

6607         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6608         { \bool_gset_false:N \g_tmpa_bool }
6609         {
6610             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6611             { \bool_gset_false:N \g_tmpa_bool }
6612         }
6613     }
6614 }
6615 }

6616 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6617 {
6618     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6619     {
6620         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6621         {
6622             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6623             { \bool_gset_false:N \g_tmpa_bool }
6624             {
6625                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6626                 { \bool_gset_false:N \g_tmpa_bool }
6627             }
6628         }
6629     }
6630 }

```

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.

```

6631 \cs_new_protected:Npn \@@_compute_corners:
6632 {
6633     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6634     { \@@_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`).

```

6635     \clist_clear:N \l_@@_corners_cells_clist
6636     \clist_map_inline:Nn \l_@@_corners_cells_clist
6637     {
6638         \str_case:nnF { ##1 }
6639         {
6640             { NW }
6641             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6642             { NE }
6643             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6644             { SW }
6645             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6646             { SE }
6647             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6648         }
6649         { \@@_error:nn { bad-corner } { ##1 } }
6650     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6651     \clist_if_empty:NF \l_@@_corners_cells_clist
6652     {

```

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.

```

6653     \tl_gput_right:Np \g_@@_aux_tl
6654     {
6655         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6656         { \l_@@_corners_cells_clist }
6657     }
6658 }
6659 }

```

```

6660 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6661 {
6662     \int_step_inline:nnn { #1 } { #3 }
6663     {
6664         \int_step_inline:nnn { #2 } { #4 }
6665         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ###1 } { } }
6666     }
6667 }

```

```

6668 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6669 {
6670     \cs_if_exist:cTF
6671     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6672     \prg_return_true:
6673     \prg_return_false:
6674 }

```

“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.

```

6675 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6676 {

```

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.

```

6677     \bool_set_false:N \l_tmpa_bool
6678     \int_zero_new:N \l_@@_last_empty_row_int
6679     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6680     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6681     {
6682         \bool_lazy_or:nnTF
6683         {
6684             \cs_if_exist_p:c
6685             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6686         }
6687         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6688         { \bool_set_true:N \l_tmpa_bool }
6689     }

```

```

6690         \bool_if:NF \l_tmpa_bool
6691         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6692     }
6693 }

```

Now, you determine the last empty cell in the row of number 1.

```

6694     \bool_set_false:N \l_tmpa_bool
6695     \int_zero_new:N \l_@@_last_empty_column_int
6696     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6697     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6698     {
6699         \bool_lazy_or:nnTF
6700         {
6701             \cs_if_exist_p:c
6702             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6703         }
6704         { \@@_if_in_block_p:nn { #1 } { ##1 } }
6705         { \bool_set_true:N \l_tmpa_bool }
6706     }
6707     \bool_if:NF \l_tmpa_bool
6708     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6709 }
6710 }

```

Now, we loop over the rows.

```

6711     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6712     {

```

We treat the row number ##1 with another loop.

```

6713         \bool_set_false:N \l_tmpa_bool
6714         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6715         {
6716             \bool_lazy_or:nnTF
6717             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6718             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6719             { \bool_set_true:N \l_tmpa_bool }
6720         }
6721         \bool_if:NF \l_tmpa_bool
6722         {
6723             \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6724             \clist_put_right:Nn
6725             \l_@@_corners_cells_clist
6726             { ##1 - #####1 }
6727             \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6728         }
6729     }
6730 }
6731 }
6732 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6733 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6734 \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.

```
6735 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6736 \keys_define:nn { nicematrix / NiceMatrixBlock }
6737 {
6738   auto-columns-width .code:n =
6739   {
6740     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6741     \dim_gzero_new:N \g_@@_max_cell_width_dim
6742     \bool_set_true:N \l_@@_auto_columns_width_bool
6743   }
6744 }

6745 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6746 {
6747   \int_gincr:N \g_@@_NiceMatrixBlock_int
6748   \dim_zero:N \l_@@_columns_width_dim
6749   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6750   \bool_if:NT \l_@@_block_auto_columns_width_bool
6751   {
6752     \cs_if_exist:cT
6753     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6754     {
6755       \dim_set:Nn \l_@@_columns_width_dim
6756       {
6757         \use:c
6758         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6759       }
6760     }
6761   }
6762 }
```

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`).

```
6763 {
6764   \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.

```
6765   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6766   {
6767     \bool_if:NT \l_@@_block_auto_columns_width_bool
6768     {
6769       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6770       \iow_shipout:Ne \@mainaux
6771       {
6772         \cs_gset:cpn
6773         { @@_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.

```
6774         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6775       }
6776       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6777     }
6778   }
6779   \ignorespacesafterend
6780 }
```

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).

```

6781 \cs_new_protected:Npn \@@_create_extra_nodes:
6782 {
6783   \bool_if:nTF \l_@@_medium_nodes_bool
6784   {
6785     \bool_if:NTF \l_@@_large_nodes_bool
6786     \@@_create_medium_and_large_nodes:
6787     \@@_create_medium_nodes:
6788   }
6789   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6790 }

```

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`.

```

6791 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6792 {
6793   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6794   {
6795     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6796     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6797     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6798     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6799   }
6800   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6801   {
6802     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6803     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6804     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6805     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6806   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6807   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6808   {
6809     \int_step_variable:nnNn
6810     \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.

```

6811     {
6812         \cs_if_exist:cT
6813         { 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`.

```

6814     {
6815         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6816         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6817         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6818         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6819         {
6820             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6821             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6822         }

```

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`.

```

6823         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6824         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6825         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6826         \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6827         {
6828             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6829             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6830         }
6831     }
6832 }
6833 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6834 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6835 {
6836     \dim_compare:nNnT
6837     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6838     {
6839         \@@_qpoint:n { row - \@@_i: - base }
6840         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6841         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6842     }
6843 }
6844 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6845 {
6846     \dim_compare:nNnT
6847     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6848     {
6849         \@@_qpoint:n { col - \@@_j: }
6850         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6851         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6852     }
6853 }
6854 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6855 \cs_new_protected:Npn \@@_create_medium_nodes:
6856 {
6857     \pgfpicture
6858     \pgfrememberpicturepositiononpagetrue
6859     \pgf@relevantforpicturesizefalse
6860     \@@_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”.

```
6861     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6862     \@@_create_nodes:
6863     \endpgfpicture
6864 }
```

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:`.

```
6865 \cs_new_protected:Npn \@@_create_large_nodes:
6866 {
6867   \pgfpicture
6868   \pgfrememberpicturepositiononpagetrue
6869   \pgf@relevantforpicturesizefalse
6870   \@@_computations_for_medium_nodes:
6871   \@@_computations_for_large_nodes:
6872   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6873   \@@_create_nodes:
6874   \endpgfpicture
6875 }

6876 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6877 {
6878   \pgfpicture
6879   \pgfrememberpicturepositiononpagetrue
6880   \pgf@relevantforpicturesizefalse
6881   \@@_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”.

```
6882     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6883     \@@_create_nodes:
6884     \@@_computations_for_large_nodes:
6885     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6886     \@@_create_nodes:
6887     \endpgfpicture
6888 }
```

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.

```
6889 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6890 {
6891   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6892   \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`.

```
6893   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6894   {
6895     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6896     {
6897       (
6898         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6899         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6900       )
6901       / 2
6902     }
}
```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6903     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6904     { l_@@_row_ \@@_i: _ min_dim }
6905   }
6906   \int_step_variable:nnNn { \c@jCol - 1 } \@@_j:
6907   {
6908     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6909     {
6910       (
6911         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6912         \dim_use:c
6913         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6914       )
6915       / 2
6916     }
6917     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6918     { l_@@_column _ \@@_j: _ max _ dim }
6919   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6920   \dim_sub:cn
6921   { l_@@_column _ 1 _ min _ dim }
6922   \l_@@_left_margin_dim
6923   \dim_add:cn
6924   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6925   \l_@@_right_margin_dim
6926 }

```

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`).

```

6927 \cs_new_protected:Npn \@@_create_nodes:
6928 {
6929   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6930   {
6931     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6932     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6933     \@@_pgf_rect_node:nnnnn
6934     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6935     { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6936     { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6937     { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6938     { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6939     \str_if_empty:NF \l_@@_name_str
6940     {
6941       \pgfnodealias
6942       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6943       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6944     }
6945   }
6946 }

```

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 .

```

6947   \seq_map_pairwise_function:NNN
6948   \g_@@_multicolumn_cells_seq
6949   \g_@@_multicolumn_sizes_seq
6950   \@@_node_for_multicolumn:nn
6951 }

```

```

6952 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6953 {
6954   \cs_set_nopar:Npn \@@_i: { #1 }
6955   \cs_set_nopar:Npn \@@_j: { #2 }
6956 }

```

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”).

```

6957 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6958 {
6959   \@@_extract_coords_values: #1 \q_stop
6960   \@@_pgf_rect_node:nnnnn
6961     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6962     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6963     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6964     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6965     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6966   \str_if_empty:NF \l_@@_name_str
6967     {
6968       \pgfnodealias
6969         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6970         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6971     }
6972 }

```

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).

```

6973 \keys_define:mn { nicematrix / Block / FirstPass }
6974 {
6975   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
6976             \bool_set_true:N \l_@@_p_block_bool ,
6977   j .value_forbidden:n = true ,
6978   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6979   l .value_forbidden:n = true ,
6980   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6981   r .value_forbidden:n = true ,
6982   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6983   c .value_forbidden:n = true ,
6984   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6985   L .value_forbidden:n = true ,
6986   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6987   R .value_forbidden:n = true ,
6988   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6989   C .value_forbidden:n = true ,
6990   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6991   t .value_forbidden:n = true ,
6992   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6993   T .value_forbidden:n = true ,
6994   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6995   b .value_forbidden:n = true ,
6996   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
6997   B .value_forbidden:n = true ,

```

```

6998   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
6999   m .value_forbidden:n = true ,
7000   v-center .meta:n = m ,
7001   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7002   p .value_forbidden:n = true ,
7003   color .code:n =
7004     \@@_color:n { #1 }
7005     \tl_set_rescan:Nnn
7006       \l_@@_draw_tl
7007       { \char_set_catcode_other:N ! }
7008       { #1 } ,
7009   color .value_required:n = true ,
7010   respect-arraystretch .code:n =
7011     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7012   respect-arraystretch .value_forbidden:n = true ,
7013 }

```

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.

```

7014 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7015 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7016 {

```

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).

```

7017   \peek_remove_spaces:n
7018   {
7019     \tl_if_blank:nTF { #2 }
7020     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7021     {
7022       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7023       \@@_Block_i_czech \@@_Block_i
7024       #2 \q_stop
7025     }
7026     { #1 } { #3 } { #4 }
7027   }
7028 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7029 \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`.

```

7030 {
7031   \char_set_catcode_active:N -
7032   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7033 }

```

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.

```

7034 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7035 {

```

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).

```

7036   \bool_lazy_or:nnTF
7037     { \tl_if_blank_p:n { #1 } }
7038     { \str_if_eq_p:ee { * } { #1 } }
7039     { \int_set:Nn \l_tmpa_int { 100 } }
7040     { \int_set:Nn \l_tmpa_int { #1 } }
7041   \bool_lazy_or:nnTF
7042     { \tl_if_blank_p:n { #2 } }
7043     { \str_if_eq_p:ee { * } { #2 } }
7044     { \int_set:Nn \l_tmpb_int { 100 } }
7045     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7046   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7047     {
7048       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7049         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7050         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7051     }
7052     { \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.

```

7053   \keys_set_known:n { nicematrix / Block / FirstPass } { #3 }
7054   \tl_set:Ne \l_tmpa_tl
7055     {
7056       { \int_use:N \c@iRow }
7057       { \int_use:N \c@jCol }
7058       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7059       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7060     }

```

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).

```

7061   \bool_set_false:N \l_tmpa_bool
7062   \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7063     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7064   \bool_case:nF
7065     {
7066       \l_tmpa_bool                               { \@@_Block_vii:eennn }
7067       \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.

```

7068       \l_@@_X_bool                               { \@@_Block_v:eennn }
7069       { \tl_if_empty_p:n { #5 } }                 { \@@_Block_v:eennn }
7070       { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7071       { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7072     }
7073     { \@@_Block_v:eennn }
7074   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7075 }

```

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.

```

7076 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7077 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7078 {
7079   \int_gincr:N \g_@@_block_box_int
7080   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7081   {
7082     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7083     {
7084       \@@_actually_diagbox:nnnnnn
7085       { \int_use:N \c@iRow }
7086       { \int_use:N \c@jCol }
7087       { \int_eval:n { \c@iRow + #1 - 1 } }
7088       { \int_eval:n { \c@jCol + #2 - 1 } }
7089       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7090       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7091     }
7092   }
7093   \box_gclear_new:c
7094   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7095   \hbox_gset:cn
7096   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7097   {

```

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}`).

```

7098   \tl_if_empty:NTF \l_@@_color_tl
7099   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7100   { \@@_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`.

```

7101   \int_compare:nNnT { #1 } = \c_one_int
7102   {
7103     \int_if_zero:nTF \c@iRow
7104     {

```

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}$

```

```

7105         \cs_set_eq:NN \Block \@@_NullBlock:
7106         \l_@@_code_for_first_row_tl
7107     }
7108     {
7109         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7110         {
7111             \cs_set_eq:NN \Block \@@_NullBlock:
7112             \l_@@_code_for_last_row_tl
7113         }
7114     }
7115     \g_@@_row_style_tl
7116 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7117     \@@_reset_arraystretch:
7118     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7119     #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.).

```

7120     \@@_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}`.

```

7121     \bool_if:NTF \l_@@_tabular_bool
7122     {
7123         \bool_lazy_all:nTF
7124         {
7125             { \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.

```

7126         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7127         { ! \g_@@_rotate_bool }
7128     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7129     {
7130         \use:e
7131         {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7132         \exp_not:N \begin { minipage }%
7133         [ \str_lowercase:o \l_@@_vpos_block_str ]
7134         { \l_@@_col_width_dim }
7135         \str_case:on \l_@@_hpos_block_str
7136         { c \centering r \raggedleft l \raggedright }
7137     }

```



```

7138         #5
7139         \end { minipage }
7140     }

```

In the other cases, we use a `{tabular}`.

```

7141     {
7142     \use:e
7143     {
7144     \exp_not:N \begin { tabular }%
7145     [ \str_lowercase:o \l_@@_vpos_block_str ]
7146     { @ { } \l_@@_hpos_block_str @ { } }
7147     }
7148     #5
7149     \end { tabular }
7150     }
7151 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7152     {
7153     \c_math_toggle_token
7154     \use:e
7155     {
7156     \exp_not:N \begin { array }%
7157     [ \str_lowercase:o \l_@@_vpos_block_str ]
7158     { @ { } \l_@@_hpos_block_str @ { } }
7159     }
7160     #5
7161     \end { array }
7162     \c_math_toggle_token
7163     }
7164 }

```

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).

```

7165     \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.

```

7166     \int_compare:nNnT { #2 } = \c_one_int
7167     {
7168     \dim_gset:Nn \g_@@_blocks_wd_dim
7169     {
7170     \dim_max:nn
7171     \g_@@_blocks_wd_dim
7172     {
7173     \box_wd:c
7174     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7175     }
7176     }
7177 }

```

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.

```

7178     \bool_lazy_and:nnT
7179     { \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.

```

7180     { \str_if_empty_p:N \l_@@_vpos_block_str }
7181     {
7182     \dim_gset:Nn \g_@@_blocks_ht_dim

```

```

7183     {
7184         \dim_max:nn
7185         \g_@@_blocks_ht_dim
7186         {
7187             \box_ht:c
7188             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7189         }
7190     }
7191 \dim_gset:Nn \g_@@_blocks_dp_dim
7192 {
7193     \dim_max:nn
7194     \g_@@_blocks_dp_dim
7195     {
7196         \box_dp:c
7197         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7198     }
7199 }
7200 }
7201 \seq_gput_right:Ne \g_@@_blocks_seq
7202 {
7203     \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.

```

7204     {
7205         \exp_not:n { #3 } ,
7206         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7207     \bool_if:NT \g_@@_rotate_bool
7208     {
7209         \bool_if:NTF \g_@@_rotate_c_bool
7210         { m }
7211         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7212     }
7213 }
7214 {
7215     \box_use_drop:c
7216     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7217 }
7218 }
7219 \bool_set_false:N \g_@@_rotate_c_bool
7220 }

```

```

7221 \cs_new:Npn \@@_adjust_hpos_rotate:
7222 {
7223     \bool_if:NT \g_@@_rotate_bool
7224     {
7225         \str_set:Ne \l_@@_hpos_block_str
7226         {
7227             \bool_if:NTF \g_@@_rotate_c_bool
7228             { c }
7229             {
7230                 \str_case:onF \l_@@_vpos_block_str
7231                 { b l B l t r T r }
7232                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7233             }
7234         }
7235     }
7236 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7237 \cs_new_protected:Npn \@@_rotate_box_of_block:
7238 {
7239   \box_grotate:cn
7240   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7241   { 90 }
7242   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7243   {
7244     \vbox_gset_top:cn
7245     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7246     {
7247       \skip_vertical:n { 0.8 ex }
7248       \box_use:c
7249       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7250     }
7251   }
7252   \bool_if:NT \g_@@_rotate_c_bool
7253   {
7254     \hbox_gset:cn
7255     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7256     {
7257       \c_math_toggle_token
7258       \vcenter
7259       {
7260         \box_use:c
7261         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7262       }
7263       \c_math_toggle_token
7264     }
7265   }
7266 }

```

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.

```

7267 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7268 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7269 {
7270   \seq_gput_right:Ne \g_@@_blocks_seq
7271   {
7272     \l_tmpa_tl
7273     { \exp_not:n { #3 } }
7274     {
7275       \bool_if:NTF \l_@@_tabular_bool
7276       {
7277         \group_begin:

```

The following command will be no-op when respect-arraystretch is in force.

```

7278   \@@_reset_arraystretch:
7279   \exp_not:n
7280   {
7281     \dim_zero:N \extrarowheight
7282     #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).

```

7283         \bool_if:NT \c_@@_testphase_table_bool
7284         { \tag_stop:n { table } }
7285         \use:e
7286         {
7287             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7288             { @ { } \l_@@_hpos_block_str @ { } }
7289         }
7290         #5
7291         \end { tabular }
7292     }
7293     \group_end:
7294 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7295     {
7296         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7297         \@@_reset_arraystretch:
7298         \exp_not:n
7299         {
7300             \dim_zero:N \extrarowheight
7301             #4
7302             \c_math_toggle_token
7303             \use:e
7304             {
7305                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7306                 { @ { } \l_@@_hpos_block_str @ { } }
7307             }
7308             #5
7309             \end { array }
7310             \c_math_toggle_token
7311         }
7312         \group_end:
7313     }
7314 }
7315 }
7316 }

```

The following macro is for the case of a \Block which uses the key p.

```

7317 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7318 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7319 {
7320     \seq_gput_right:Ne \g_@@_blocks_seq
7321     {
7322         \l_tmpa_tl
7323         { \exp_not:n { #3 } }
7324         {
7325             \group_begin:
7326             \exp_not:n { #4 #5 }
7327             \group_end:
7328         }
7329     }
7330 }

```

The following macro is for the case of a \Block which uses the key p.

```

7331 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7332 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7333 {
7334     \seq_gput_right:Ne \g_@@_blocks_seq
7335     {

```

```

7336     \l_tmpa_tl
7337     { \exp_not:n { #3 } }
7338     { \exp_not:n { #4 #5 } }
7339   }
7340 }

```

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).

```

7341 \keys_define:nn { nicematrix / Block / SecondPass }
7342 {
7343   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7344   ampersand-in-blocks .default:n = true ,
7345   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7346   tikz .code:n =
7347     \IfPackageLoadedTF { tikz }
7348     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7349     { \@@_error:n { tikz~key~without~tikz } } ,
7350   tikz .value_required:n = true ,
7351   fill .code:n =
7352     \tl_set_rescan:Nnn
7353     \l_@@_fill_tl
7354     { \char_set_catcode_other:N ! }
7355     { #1 } ,
7356   fill .value_required:n = true ,
7357   opacity .tl_set:N = \l_@@_opacity_tl ,
7358   opacity .value_required:n = true ,
7359   draw .code:n =
7360     \tl_set_rescan:Nnn
7361     \l_@@_draw_tl
7362     { \char_set_catcode_other:N ! }
7363     { #1 } ,
7364   draw .default:n = default ,
7365   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7366   rounded-corners .default:n = 4 pt ,
7367   color .code:n =
7368     \@@_color:n { #1 }
7369     \tl_set_rescan:Nnn
7370     \l_@@_draw_tl
7371     { \char_set_catcode_other:N ! }
7372     { #1 } ,
7373   borders .clist_set:N = \l_@@_borders_clist ,
7374   borders .value_required:n = true ,
7375   hvlines .meta:n = { vlines , hlines } ,
7376   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7377   vlines .default:n = true ,
7378   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7379   hlines .default:n = true ,
7380   line-width .dim_set:N = \l_@@_line_width_dim ,
7381   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7382   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7383     \bool_set_true:N \l_@@_p_block_bool ,
7384   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7385   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7386   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7387   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7388     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7389   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7390     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,

```

```

7391 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7392         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7393 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7394 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7395 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7396 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7397 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7398 m .value_forbidden:n = true ,
7399 v-center .meta:n = m ,
7400 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7401 p .value_forbidden:n = true ,
7402 name .tl_set:N = \l_@@_block_name_str ,
7403 name .value_required:n = true ,
7404 name .initial:n = ,
7405 respect-arraystretch .code:n =
7406     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7407 respect-arraystretch .value_forbidden:n = true ,
7408 transparent .bool_set:N = \l_@@_transparent_bool ,
7409 transparent .default:n = true ,
7410 transparent .initial:n = false ,
7411 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7412 }

```

The command `\@@_draw_blocks:` will draw all the blocks. This command is used after the construction of the array. We have to revert to a clean version of `\ialign` because there may be tabulars in the `\Block` instructions that will be composed now.

```

7413 \cs_new_protected:Npn \@@_draw_blocks:
7414 {
7415     \bool_if:NTF \c_@@_recent_array_bool
7416     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7417     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7418     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7419 }
7420 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7421 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7422 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7423     \int_zero_new:N \l_@@_last_row_int
7424     \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

7425     \int_compare:nNnTF { #3 } > { 99 }
7426     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7427     { \int_set:Nn \l_@@_last_row_int { #3 } }
7428     \int_compare:nNnTF { #4 } > { 99 }
7429     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7430     { \int_set:Nn \l_@@_last_col_int { #4 } }
7431     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7432     {
7433         \bool_lazy_and:nnTF
7434         \l_@@_preamble_bool
7435         {
7436             \int_compare_p:n
7437             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7438         }

```

```

7439     {
7440     \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7441     \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7442     \@@_msg_redirect_name:nn { columns-not-used } { none }
7443     }
7444     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7445   }
7446   {
7447   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7448   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7449   {
7450     \@@_Block_v:nneenn
7451     { #1 }
7452     { #2 }
7453     { \int_use:N \l_@@_last_row_int }
7454     { \int_use:N \l_@@_last_col_int }
7455     { #5 }
7456     { #6 }
7457   }
7458 }
7459 }

```

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

```

7460 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7461 {

```

The group is for the keys.

```

7462   \group_begin:
7463   \int_compare:nNnT { #1 } = { #3 }
7464   { \str_set:Nn \l_@@_vpos_block_str { t } }
7465   \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`.

```

7466   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7467   \bool_lazy_and:nnT
7468   \l_@@_vlines_block_bool
7469   { ! \l_@@_ampersand_bool }
7470   {
7471     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7472     {
7473       \@@_vlines_block:nnn
7474       { \exp_not:n { #5 } }
7475       { #1 - #2 }
7476       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7477     }
7478   }
7479   \bool_if:NT \l_@@_hlines_block_bool
7480   {
7481     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7482     {
7483       \@@_hlines_block:nnn
7484       { \exp_not:n { #5 } }
7485       { #1 - #2 }
7486       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7487     }
7488   }
7489   \bool_if:NF \l_@@_transparent_bool
7490   {
7491     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7492     {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7493     \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7494     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7495   }
7496 }

```

```

7497 \tl_if_empty:NF \l_@@_draw_tl
7498 {
7499   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7500   { \@@_error:n { hlines~with~color } }
7501   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7502   {
7503     \@@_stroke_block:nnn

```

#5 are the options

```

7504     { \exp_not:n { #5 } }
7505     { #1 - #2 }
7506     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7507   }
7508   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7509   { { #1 } { #2 } { #3 } { #4 } }
7510 }

```

```

7511 \clist_if_empty:NF \l_@@_borders_clist
7512 {
7513   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7514   {
7515     \@@_stroke_borders_block:nnn
7516     { \exp_not:n { #5 } }
7517     { #1 - #2 }
7518     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7519   }
7520 }

```

```

7521 \tl_if_empty:NF \l_@@_fill_tl
7522 {
7523   \tl_if_empty:NF \l_@@_opacity_tl
7524   {
7525     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7526       {
7527         \tl_set:Ne \l_@@_fill_tl
7528         {
7529           [ opacity = \l_@@_opacity_tl ,
7530           \tl_tail:o \l_@@_fill_tl
7531         }
7532       }
7533     {
7534       \tl_set:Ne \l_@@_fill_tl
7535       { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7536     }
7537   }
7538   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7539   {
7540     \exp_not:N \roundedrectanglecolor
7541     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7542       { \l_@@_fill_tl }
7543       { { \l_@@_fill_tl } }
7544     { #1 - #2 }
7545     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7546     { \dim_use:N \l_@@_rounded_corners_dim }
7547   }
7548 }

```



```

7549 \seq_if_empty:NF \l_@@_tikz_seq
7550 {
7551   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7552   {
7553     \@@_block_tikz:nnnnn
7554     { \seq_use:Nn \l_@@_tikz_seq { , } }
7555     { #1 }
7556     { #2 }
7557     { \int_use:N \l_@@_last_row_int }
7558     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7559   }
7560 }

7561 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7562 {
7563   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7564   {
7565     \@@_actually_diagbox:nnnnnn
7566     { #1 }
7567     { #2 }
7568     { \int_use:N \l_@@_last_row_int }
7569     { \int_use:N \l_@@_last_col_int }
7570     { \exp_not:n { ##1 } }
7571     { \exp_not:n { ##2 } }
7572   }
7573 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & & two & \\\
three & & four & five & \\\
six & & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block	one
three	two
six	five
	eight

The construction of the node corresponding to the merged cells.

```

7574 \pgfpicture
7575 \pgfrememberpicturepositiononpagetrue
7576 \pgf@relevantforpicturesizefalse
7577 \@@_qpoint:n { row - #1 }
7578 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7579 \@@_qpoint:n { col - #2 }
7580 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7581 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7582 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7583 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7584 \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.

```

7585 \@@_pgf_rect_node:nnnnn
7586   { \@@_env: - #1 - #2 - block }
7587   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7588 \str_if_empty:NF \l_@@_block_name_str
7589   {
7590     \pgfnodealias
7591     { \@@_env: - \l_@@_block_name_str }
7592     { \@@_env: - #1 - #2 - block }
7593     \str_if_empty:NF \l_@@_name_str
7594     {
7595       \pgfnodealias
7596       { \l_@@_name_str - \l_@@_block_name_str }
7597       { \@@_env: - #1 - #2 - block }
7598     }
7599   }

```

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.

```

7600 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7601   {
7602     \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.

```

7603     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7604     {

```

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.

```

7605       \cs_if_exist:cT
7606       { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7607       {
7608         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7609         {
7610           \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7611           \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7612         }
7613       }
7614     }

```

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.

```

7615     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7616     {
7617       \@@_qpoint:n { col - #2 }
7618       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7619     }
7620     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7621     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7622     {
7623       \cs_if_exist:cT
7624       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7625       {
7626         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7627         {
7628           \pgfpointanchor
7629           { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7630           { east }

```

```

7631         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7632     }
7633 }
7634 }
7635 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7636 {
7637     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7638     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7639 }
7640 \@@_pgf_rect_node:nnnnn
7641 { \@@_env: - #1 - #2 - block - short }
7642 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7643 }

```

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.

```

7644 \bool_if:NT \l_@@_medium_nodes_bool
7645 {
7646     \@@_pgf_rect_node:nnn
7647     { \@@_env: - #1 - #2 - block - medium }
7648     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7649     {
7650         \pgfpointanchor
7651         { \@@_env:
7652             - \int_use:N \l_@@_last_row_int
7653             - \int_use:N \l_@@_last_col_int - medium
7654         }
7655         { south~east }
7656     }
7657 }
7658 \endpgfpicture

```

```

7659 \bool_if:NTF \l_@@_ampersand_bool
7660 {
7661     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7662     \int_zero_new:N \l_@@_split_int
7663     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7664     \pgfpicture
7665     \pgfrememberpicturepositiononpagetrue
7666     \pgf@relevantforpicturesizefalse
7667
7668     \@@_qpoint:n { row - #1 }
7669     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7670     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7671     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7672     \@@_qpoint:n { col - #2 }
7673     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7674     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7675     \dim_set:Nn \l_tmpb_dim
7676     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7677     \bool_lazy_or:nnT
7678     \l_@@_vlines_block_bool
7679     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7680     {
7681         \int_step_inline:nn { \l_@@_split_int - 1 }
7682         {
7683             \pgfpathmoveto
7684             {
7685                 \pgfpoint
7686                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7687                 \l_@@_tmpc_dim
7688             }
7689             \pgfpathlineto

```

```

7690         {
7691             \pgfpoint
7692             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7693             \l_@@_tmpd_dim
7694         }
7695         \CT@arc@
7696         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7697         \pgfsetrectcap
7698         \pgfusepathqstroke
7699     }
7700 }
7701 \@@_qpoint:n { row - #1 - base }
7702 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7703 \int_step_inline:nn \l_@@_split_int
7704 {
7705     \group_begin:
7706     \dim_set:Nn \col@sep
7707     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7708     \pgftransformshift
7709     {
7710         \pgfpoint
7711         {
7712             \l_tmpa_dim + ##1 \l_tmpb_dim -
7713             \str_case:on \l_@@_hpos_block_str
7714             {
7715                 l { \l_tmpb_dim + \col@sep }
7716                 c { 0.5 \l_tmpb_dim }
7717                 r { \col@sep }
7718             }
7719         }
7720         { \l_@@_tmpc_dim }
7721     }
7722     \pgfset { inner~sep = \c_zero_dim }
7723     \pgfnode
7724     { rectangle }
7725     {
7726         \str_case:on \l_@@_hpos_block_str
7727         {
7728             c { base }
7729             l { base-west }
7730             r { base-east }
7731         }
7732     }
7733     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7734     \group_end:
7735 }
7736 \endpgfpicture
7737 }

```

Now the case where there is no ampersand & in the content of the block.

```

7738 {
7739     \bool_if:NTF \l_@@_p_block_bool
7740     {

```

When the final user has used the key p, we have to compute the width.

```

7741     \pgfpicture
7742     \pgfrememberpicturepositiononpagetrue
7743     \pgf@relevantforpicturesizefalse
7744     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7745     {
7746         \@@_qpoint:n { col - #2 }
7747         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7748         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7749     }

```

```

7750     {
7751         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7752         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7753         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7754     }
7755     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7756 \endpgfpicture
7757 \hbox_set:Nn \l_@@_cell_box
7758 {
7759     \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7760         { \g_tmpb_dim }
7761         \str_case:on \l_@@_hpos_block_str
7762             { c \centering r \raggedleft l \raggedright j { } }
7763         #6
7764     \end { minipage }
7765 }
7766 }
7767 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7768 \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.

```

7769 \pgfpicture
7770 \pgfrememberpicturepositiononpagetrue
7771 \pgf@relevantforpicturesizefalse
7772 \bool_lazy_any:nTF
7773 {
7774     { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7775     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7776     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7777     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7778 }
7779 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7780     \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`.

```

7781     \bool_if:nT \g_@@_last_col_found_bool
7782     {
7783         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7784         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7785     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7786     \tl_set:Ne \l_tmpa_tl
7787     {
7788         \str_case:on \l_@@_vpos_block_str
7789         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7790         { } { % added 2024-06-29
7791             \str_case:on \l_@@_hpos_block_str
7792             {
7793                 c { center }
7794                 l { west }
7795                 r { east }
7796                 j { center }
7797             }
7798         }
7799     c {
7800         \str_case:on \l_@@_hpos_block_str

```

```

7801         {
7802             c { center }
7803             l { west }
7804             r { east }
7805             j { center }
7806         }
7807     }
7808 }
7809 T {
7810     \str_case:on \l_@@_hpos_block_str
7811     {
7812         c { north }
7813         l { north-west }
7814         r { north-east }
7815         j { north }
7816     }
7817 }
7818 }
7819 B {
7820     \str_case:on \l_@@_hpos_block_str
7821     {
7822         c { south }
7823         l { south-west }
7824         r { south-east }
7825         j { south }
7826     }
7827 }
7828 }
7829 }
7830 }
7831 \pgftransformshift
7832 {
7833     \pgfpointanchor
7834     {
7835         \@@_env: - #1 - #2 - block
7836         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7837     }
7838     { \l_tmpa_tl }
7839 }
7840 \pgfset { inner-sep = \c_zero_dim }
7841 \pgfnode
7842 { rectangle }
7843 { \l_tmpa_tl }
7844 { \box_use_drop:N \l_@@_cell_box } { } { }
7845 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7846 {
7847     \pgfextracty \l_tmpa_dim
7848     {
7849         \@@_qpoint:n
7850         {
7851             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7852             - base
7853         }
7854     }
7855     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\pgf@x$) the x -value of the center of the block.

```

7856 \pgfpointanchor
7857 {
7858     \@@_env: - #1 - #2 - block
7859     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }

```

```

7860     }
7861     {
7862         \str_case:on \l_@@_hpos_block_str
7863         {
7864             c { center }
7865             l { west }
7866             r { east }
7867             j { center }
7868         }
7869     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7870         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7871         \pgfset { inner~sep = \c_zero_dim }
7872         \pgfnode
7873         { rectangle }
7874         {
7875             \str_case:on \l_@@_hpos_block_str
7876             {
7877                 c { base }
7878                 l { base~west }
7879                 r { base~east }
7880                 j { base }
7881             }
7882         }
7883         { \box_use_drop:N \l_@@_cell_box } { } { }
7884     }
7885     \endpgfpicture
7886 }
7887 \group_end:
7888 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

7889 \cs_set_protected:Npn \@@_fill:nnnn #1 #2 #3 #4 #5
7890 {
7891     \pgfpicture
7892     \pgfrememberpicturepositiononpagetrue
7893     \pgf@relevantforpicturesizefalse
7894     \pgfpathrectanglecorners
7895     { \pgfpoint { #2 } { #3 } }
7896     { \pgfpoint { #4 } { #5 } }
7897     \pgfsetfillcolor { #1 }
7898     \pgfusepath { fill }
7899     \endpgfpicture
7900 }

```

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).

```

7901 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7902 {
7903     \group_begin:
7904     \tl_clear:N \l_@@_draw_tl
7905     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7906     \keys_set_known:nm { nicematrix / BlockStroke } { #1 }
7907     \pgfpicture
7908     \pgfrememberpicturepositiononpagetrue
7909     \pgf@relevantforpicturesizefalse
7910     \tl_if_empty:NF \l_@@_draw_tl
7911     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7912     \tl_if_eq:NnTF \l_@@_draw_tl { default }
7913         { \CT@arc@ }
7914         { \@@_color:o \l_@@_draw_tl }
7915     }
7916 \pgfsetcornersarced
7917 {
7918     \pgfpoint
7919     { \l_@@_rounded_corners_dim }
7920     { \l_@@_rounded_corners_dim }
7921 }
7922 \@@_cut_on_hyphen:w #2 \q_stop
7923 \int_compare:nNnF \l_tmpa_tl > \c@iRow
7924 {
7925     \int_compare:nNnF \l_tmpb_tl > \c@jCol
7926     {
7927         \@@_qpoint:n { row - \l_tmpa_tl }
7928         \dim_set_eq:NN \l_tmpb_dim \pgf@y
7929         \@@_qpoint:n { col - \l_tmpb_tl }
7930         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7931         \@@_cut_on_hyphen:w #3 \q_stop
7932         \int_compare:nNnT \l_tmpa_tl > \c@iRow
7933             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7934         \int_compare:nNnT \l_tmpb_tl > \c@jCol
7935             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7936         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7937         \dim_set_eq:NN \l_tmpa_dim \pgf@y
7938         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7939         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7940         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7941         \pgfpathrectanglecorners
7942             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7943             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7944         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7945             { \pgfusepathqstroke }
7946             { \pgfusepath { stroke } }
7947     }
7948 }
7949 \endpgfpicture
7950 \group_end:
7951 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7952 \keys_define:nn { nicematrix / BlockStroke }
7953 {
7954     color .tl_set:N = \l_@@_draw_tl ,
7955     draw .code:n =
7956         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7957     draw .default:n = default ,
7958     line-width .dim_set:N = \l_@@_line_width_dim ,
7959     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7960     rounded-corners .default:n = 4 pt
7961 }

```

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).

```

7962 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7963 {
7964     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7965     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7966     \@@_cut_on_hyphen:w #2 \q_stop

```



```

7967 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7968 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7969 \@@_cut_on_hyphen:w #3 \q_stop
7970 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7971 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7972 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7973 {
7974   \use:e
7975   {
7976     \@@_vline:n
7977     {
7978       position = ##1 ,
7979       start = \l_@@_tmpc_tl ,
7980       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7981       total-width = \dim_use:N \l_@@_line_width_dim
7982     }
7983   }
7984 }
7985 }
7986 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7987 {
7988   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7989   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7990   \@@_cut_on_hyphen:w #2 \q_stop
7991   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7992   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7993   \@@_cut_on_hyphen:w #3 \q_stop
7994   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7995   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7996   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7997   {
7998     \use:e
7999     {
8000       \@@_hline:n
8001       {
8002         position = ##1 ,
8003         start = \l_@@_tmpd_tl ,
8004         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8005         total-width = \dim_use:N \l_@@_line_width_dim
8006       }
8007     }
8008   }
8009 }

```

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).

```

8010 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8011 {
8012   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8013   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8014   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8015     { \@@_error:n { borders~forbidden } }
8016     {
8017       \tl_clear_new:N \l_@@_borders_tikz_tl
8018       \keys_set:no
8019         { nicematrix / OnlyForTikzInBorders }
8020       \l_@@_borders_clist
8021       \@@_cut_on_hyphen:w #2 \q_stop
8022       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8023       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8024       \@@_cut_on_hyphen:w #3 \q_stop
8025       \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }

```

```

8026     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8027     \@@_stroke_borders_block_i:
8028   }
8029 }

8030 \hook_gput_code:nnn { begindocument } { . }
8031 {
8032   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8033   {
8034     \c_@@_pgfortikzpicture_tl
8035     \@@_stroke_borders_block_ii:
8036     \c_@@_endpgfortikzpicture_tl
8037   }
8038 }

8039 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8040 {
8041   \pgfrememberpicturepositiononpagetrue
8042   \pgf@relevantforpicturesizefalse
8043   \CT@arc@
8044   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8045   \clist_if_in:NnT \l_@@_borders_clist { right }
8046   { \@@_stroke_vertical:n \l_tmpb_tl }
8047   \clist_if_in:NnT \l_@@_borders_clist { left }
8048   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8049   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8050   { \@@_stroke_horizontal:n \l_tmpa_tl }
8051   \clist_if_in:NnT \l_@@_borders_clist { top }
8052   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8053 }

8054 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8055 {
8056   tikz .code:n =
8057     \cs_if_exist:NTF \tikzpicture
8058     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8059     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8060   tikz .value_required:n = true ,
8061   top .code:n = ,
8062   bottom .code:n = ,
8063   left .code:n = ,
8064   right .code:n = ,
8065   unknown .code:n = \@@_error:n { bad~border }
8066 }

```

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).

```

8067 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8068 {
8069   \@@_qpoint:n \l_@@_tmpc_tl
8070   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8071   \@@_qpoint:n \l_tmpa_tl
8072   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8073   \@@_qpoint:n { #1 }
8074   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8075   {
8076     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8077     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8078     \pgfusepathqstroke
8079   }
8080   {
8081     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8082     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8083   }
8084 }

```

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).

```

8085 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8086 {
8087   \@@_qpoint:n \l_@@_tmpd_tl
8088   \clist_if_in:NnTF \l_@@_borders_clist { left }
8089     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8090     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8091   \@@_qpoint:n \l_tmpb_tl
8092   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8093   \@@_qpoint:n { #1 }
8094   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8095     {
8096       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8097       \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8098       \pgfusepathqstroke
8099     }
8100     {
8101       \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8102         ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8103     }
8104 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8105 \keys_define:nn { nicematrix / BlockBorders }
8106 {
8107   borders .clist_set:N = \l_@@_borders_clist ,
8108   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8109   rounded-corners .default:n = 4 pt ,
8110   line-width .dim_set:N = \l_@@_line_width_dim
8111 }

```

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.

```

8112 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8113 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8114 {
8115   \begin { tikzpicture }
8116   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8117   \clist_map_inline:nn { #1 }
8118   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8119     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8120     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8121     (
8122       [
8123         xshift = \dim_use:N \l_@@_offset_dim ,
8124         yshift = - \dim_use:N \l_@@_offset_dim
8125       ]
8126       #2 -| #3
8127     )
8128     rectangle
8129     (
8130     [

```

```

8131         xshift = - \dim_use:N \l_@@_offset_dim ,
8132         yshift = \dim_use:N \l_@@_offset_dim
8133     ]
8134     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8135 ) ;
8136 }
8137 \end { tikzpicture }
8138 }

8139 \keys_define:nn { nicematrix / SpecialOffset }
8140 { 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.

```

8141 \cs_new_protected:Npn \@@_NullBlock:
8142 { \@@_collect_options:n { \@@_NullBlock_i: } }
8143 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8144 { }

```

27 How to draw the dotted lines transparently

```

8145 \cs_set_protected:Npn \@@_renew_matrix:
8146 {
8147   \RenewDocumentEnvironment { pmatrix } { }
8148   { \pNiceMatrix }
8149   { \endpNiceMatrix }
8150   \RenewDocumentEnvironment { vmatrix } { }
8151   { \vNiceMatrix }
8152   { \endvNiceMatrix }
8153   \RenewDocumentEnvironment { Vmatrix } { }
8154   { \VNiceMatrix }
8155   { \endVNiceMatrix }
8156   \RenewDocumentEnvironment { bmatrix } { }
8157   { \bNiceMatrix }
8158   { \endbNiceMatrix }
8159   \RenewDocumentEnvironment { Bmatrix } { }
8160   { \BNiceMatrix }
8161   { \endBNiceMatrix }
8162 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8163 \keys_define:nn { nicematrix / Auto }
8164 {
8165   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8166   columns-type .value_required:n = true ,
8167   l .meta:n = { columns-type = l } ,
8168   r .meta:n = { columns-type = r } ,
8169   c .meta:n = { columns-type = c } ,
8170   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8171   delimiters / color .value_required:n = true ,
8172   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8173   delimiters / max-width .default:n = true ,
8174   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8175   delimiters .value_required:n = true ,

```

```

8176     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8177     rounded-corners .default:n = 4 pt
8178 }

8179 \NewDocumentCommand \AutoNiceMatrixWithDelims
8180 { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8181 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8182 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8183 {

```

The group is for the protection of the keys.

```

8184     \group_begin:
8185     \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8186     \use:e
8187     {
8188         \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8189         { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8190         [ \exp_not:o \l_tmpa_tl ]
8191     }
8192     \int_if_zero:nT \l_@@_first_row_int
8193     {
8194         \int_if_zero:nT \l_@@_first_col_int { & }
8195         \prg_replicate:nn { #4 - 1 } { & }
8196         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8197     }
8198     \prg_replicate:mn { #3 }
8199     {
8200         \int_if_zero:nT \l_@@_first_col_int { & }

```

We put { } before #6 to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```

8201         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8202         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8203     }
8204     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8205     {
8206         \int_if_zero:nT \l_@@_first_col_int { & }
8207         \prg_replicate:nn { #4 - 1 } { & }
8208         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8209     }
8210     \end { NiceArrayWithDelims }
8211     \group_end:
8212 }

8213 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8214 {
8215     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8216     {
8217         \bool_gset_true:N \g_@@_delims_bool
8218         \str_gset_Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8219         \AutoNiceMatrixWithDelims { #2 } { #3 }
8220     }
8221 }

8222 \@@_define_com:nnn p ( )
8223 \@@_define_com:nnn b [ ]
8224 \@@_define_com:nnn v | |
8225 \@@_define_com:nnn V \! \!
8226 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8227 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8228 {

```

```

8229 \group_begin:
8230 \bool_gset_false:N \g_@@_delims_bool
8231 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8232 \group_end:
8233 }

```

29 The redefinition of the command `\dotfill`

```

8234 \cs_set_eq:NN \@@_old_dotfill \dotfill
8235 \cs_new_protected:Npn \@@_dotfill:
8236 {

```

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}`).

```

8237 \@@_old_dotfill
8238 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8239 }

```

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`.

```

8240 \cs_new_protected:Npn \@@_dotfill_i:
8241 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8242 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8243 {
8244 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8245 {
8246 \@@_actually_diagbox:nnnnnn
8247 { \int_use:N \c@iRow }
8248 { \int_use:N \c@jCol }
8249 { \int_use:N \c@iRow }
8250 { \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.

```

8251 { \g_@@_row_style_tl \exp_not:n { #1 } }
8252 { \g_@@_row_style_tl \exp_not:n { #2 } }
8253 }

```

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`.

```

8254 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8255 {
8256 { \int_use:N \c@iRow }
8257 { \int_use:N \c@jCol }
8258 { \int_use:N \c@iRow }
8259 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8260     { }
8261   }
8262 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8263 \cs_new_protected:Npn \@@_actually_diagbox:nnnnn #1 #2 #3 #4 #5 #6
8264 {
8265   \pgfpicture
8266   \pgf@relevantforpicturesizefalse
8267   \pgfrememberpicturepositiononpagetrue
8268   \@@_qpoint:n { row - #1 }
8269   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8270   \@@_qpoint:n { col - #2 }
8271   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8272   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8273   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8274   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8275   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8276   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8277   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8278   {

```

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.

```

8279     \CT@arc@
8280     \pgfsetroundcap
8281     \pgfusepathqstroke
8282   }
8283   \pgfset { inner~sep = 1 pt }
8284   \pgfscope
8285   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8286   \pgfnode { rectangle } { south~west }
8287   {
8288     \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8289     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8290     \end { minipage }
8291   }
8292   { }
8293   { }
8294   \endpgfscope
8295   \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8296   \pgfnode { rectangle } { north~east }
8297   {
8298     \begin { minipage } { 20 cm }
8299     \raggedleft
8300     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8301     \end { minipage }
8302   }
8303   { }
8304   { }
8305   \endpgfpicture
8306 }

```

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`.

```
8307 \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 `\`.

```
8308 \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`.

```
8309 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8310 {
8311   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8312   \@@_CodeAfter_iv:n
8313 }
```

We catch the argument of the command `\end` (in `#1`).

```
8314 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8315 {
```

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.

```
8316   \str_if_eq:eeTF \@currenvir { #1 }
8317   { \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`.

```
8318   {
8319     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8320     \@@_CodeAfter_ii:n
8321   }
8322 }
```

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.

```
8323 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8324 {
8325   \pgfpicture
8326   \pgfrememberpicturepositiononpagetrue
8327   \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.

```

8328   \@@_qpoint:n { row - 1 }
8329   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8330   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8331   \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).

```

8332   \bool_if:nTF { #3 }
8333     { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8334     { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8335   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8336     {
8337       \cs_if_exist:cT
8338         { pgf @ sh @ ns @ \@@_env: - #1 - #2 }
8339         {
8340           \pgfpointanchor
8341             { \@@_env: - #1 - #2 }
8342             { \bool_if:nTF { #3 } { west } { east } }
8343           \dim_set:Nn \l_tmpa_dim
8344             { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8345         }
8346     }

```

Now we can put the delimiter with a node of PGF.

```

8347   \pgfset { inner~sep = \c_zero_dim }
8348   \dim_zero:N \nulldelimiterspace
8349   \pgftransformshift
8350     {
8351       \pgfpoint
8352         { \l_tmpa_dim }
8353         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8354     }
8355   \pgfnode
8356     { rectangle }
8357     { \bool_if:nTF { #3 } { east } { west } }
8358     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8359     \nullfont
8360     \c_math_toggle_token
8361     \@@_color:o \l_@@_delimiters_color_tl
8362     \bool_if:nTF { #3 } { \left #1 } { \left . }
8363     \vcenter
8364     {
8365       \nullfont
8366       \hrule \@height
8367         \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8368         \@depth \c_zero_dim
8369         \@width \c_zero_dim
8370     }
8371     \bool_if:nTF { #3 } { \right . } { \right #1 }
8372     \c_math_toggle_token
8373   }
8374   { }
8375   { }
8376 \endpgfpicture
8377 }

```

33 The command `\SubMatrix`

```

8378 \keys_define:nn { nicematrix / sub-matrix }
8379 {
8380   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8381   extra-height .value_required:n = true ,
8382   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8383   left-xshift .value_required:n = true ,
8384   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8385   right-xshift .value_required:n = true ,
8386   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8387   xshift .value_required:n = true ,
8388   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8389   delimiters / color .value_required:n = true ,
8390   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8391   slim .default:n = true ,
8392   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8393   hlines .default:n = all ,
8394   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8395   vlines .default:n = all ,
8396   hvlines .meta:n = { hlines, vlines } ,
8397   hvlines .value_forbidden:n = true
8398 }
8399 \keys_define:nn { nicematrix }
8400 {
8401   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8402   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8403   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8404   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8405 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8406 \keys_define:nn { nicematrix / SubMatrix }
8407 {
8408   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8409   delimiters / color .value_required:n = true ,
8410   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8411   hlines .default:n = all ,
8412   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8413   vlines .default:n = all ,
8414   hvlines .meta:n = { hlines, vlines } ,
8415   hvlines .value_forbidden:n = true ,
8416   name .code:n =
8417     \tl_if_empty:nTF { #1 }
8418     { \@@_error:n { Invalid-name } }
8419     {
8420       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8421       {
8422         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8423         { \@@_error:nn { Duplicate-name~for~SubMatrix } { #1 } }
8424         {
8425           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8426           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8427         }
8428       }
8429       { \@@_error:n { Invalid-name } }
8430     } ,
8431   name .value_required:n = true ,
8432   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8433   rules .value_required:n = true ,
8434   code .tl_set:N = \l_@@_code_tl ,

```

```

8435     code .value_required:n = true ,
8436     unknown .code:n = \@_error:n { Unknown~key~for~SubMatrix }
8437 }

8438 \NewDocumentCommand \@_SubMatrix_in_code_before { m m m m ! 0 { } }
8439 {
8440   \peek_remove_spaces:n
8441   {
8442     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8443     {
8444       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8445       [
8446         delimiters / color = \l_@@_delimiters_color_tl ,
8447         hlines = \l_@@_submatrix_hlines_clist ,
8448         vlines = \l_@@_submatrix_vlines_clist ,
8449         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8450         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8451         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8452         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8453         #5
8454       ]
8455     }
8456     \@_SubMatrix_in_code_before_i { #2 } { #3 }
8457   }
8458 }

8459 \NewDocumentCommand \@_SubMatrix_in_code_before_i
8460 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8461 { \@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8462 \cs_new_protected:Npn \@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8463 {
8464   \seq_gput_right:Ne \g_@@_submatrix_seq
8465   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8466     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8467     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8468     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8469     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8470   }
8471 }

```

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`.

```

8472 \hook_gput_code:nnn { begindocument } { . }
8473 {
8474   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8475   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8476 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8477 {
8478   \peek_remove_spaces:n
8479   {
8480     \@@_sub_matrix:nnnnnnn
8481     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8482   }
8483 }
8484 }

```

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).

```

8485 \NewDocumentCommand \@@_compute_i_j:nn
8486 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8487 { \@@_compute_i_j:nnnn #1 #2 }

8488 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8489 {
8490   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8491   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8492   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8493   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8494   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8495     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8496   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8497     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8498   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8499     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8500   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8501     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8502 }

8503 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8504 {
8505   \group_begin:

```

The four following token lists correspond to the position of the \backslash SubMatrix.

```

8506 \@@_compute_i_j:nn { #2 } { #3 }
8507 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8508   { \cs_set_nopar:Npn \arraystretch { 1 } }
8509 \bool_lazy_or:nnTF
8510   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8511   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8512   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8513   {
8514     \str_clear_new:N \l_@@_submatrix_name_str
8515     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8516     \pgfpicture
8517     \pgfrememberpicturepositiononpagetrue
8518     \pgf@relevantforpicturesizefalse
8519     \pgfset { inner~sep = \c_zero_dim }
8520     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8521     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \backslash int_step_inline:nnn is provided by curriification.

```

8522 \bool_if:NTF \l_@@_submatrix_slim_bool
8523   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8524   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8525   {
8526     \cs_if_exist:cT
8527       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8528       {
8529         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8530         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim

```

```

8531         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8532     }
8533     \cs_if_exist:cT
8534     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8535     {
8536         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8537         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8538         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8539     }
8540 }
8541 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8542 { \@@_error:nn { Impossible-delimiter } { left } }
8543 {
8544     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8545     { \@@_error:nn { Impossible-delimiter } { right } }
8546     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8547 }
8548 \endpgfpicture
8549 }
8550 \group_end:
8551 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8552 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8553 {
8554     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8555     \dim_set:Nn \l_@@_y_initial_dim
8556     {
8557         \fp_to_dim:n
8558         {
8559             \pgf@y
8560             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8561         }
8562     }
8563     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8564     \dim_set:Nn \l_@@_y_final_dim
8565     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8566     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8567     {
8568         \cs_if_exist:cT
8569         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8570         {
8571             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8572             \dim_set:Nn \l_@@_y_initial_dim
8573             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8574         }
8575         \cs_if_exist:cT
8576         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8577         {
8578             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8579             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8580             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8581         }
8582     }
8583     \dim_set:Nn \l_tmpa_dim
8584     {
8585         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8586         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8587     }
8588     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8589   \group_begin:
8590   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8591   \@@_set_CT@arc@:o \l_@@_rules_color_tl
8592   \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`.

```

8593   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8594   {
8595     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8596     {
8597       \int_compare:nNnT
8598         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8599       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8600         \@@_qpoint:n { col - ##1 }
8601         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8602         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8603         \pgfusepathqstroke
8604       }
8605     }
8606   }

```

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.

```

8607   \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8608   { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8609   { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8610   {
8611     \bool_lazy_and:nnTF
8612     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8613     {
8614       \int_compare_p:nNn
8615       { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8616     {
8617       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8618       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8619       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8620       \pgfusepathqstroke
8621     }
8622     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8623   }

```

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.

```

8624   \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8625   { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8626   { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8627   {
8628     \bool_lazy_and:nnTF
8629     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8630     {
8631       \int_compare_p:nNn
8632       { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8633     {
8634       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8635   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8636   \dim_set:Nn \l_tmpa_dim

```

```

8637         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8638     \str_case:nn { #1 }
8639     {
8640         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8641         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8642         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8643         }
8644     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8645     \dim_set:Nn \l_tmpb_dim
8646     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8647     \str_case:nn { #2 }
8648     {
8649         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8650         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8651         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8652     }
8653     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8654     \pgfusepathqstroke
8655     \group_end:
8656 }
8657 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8658 }

```

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).

```

8659     \str_if_empty:NF \l_@@_submatrix_name_str
8660     {
8661         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8662         \l_@@_x_initial_dim \l_@@_y_initial_dim
8663         \l_@@_x_final_dim \l_@@_y_final_dim
8664     }
8665     \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`.

```

8666     \begin { pgfscope }
8667     \pgftransformshift
8668     {
8669         \pgfpoint
8670         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8671         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8672     }
8673     \str_if_empty:NTF \l_@@_submatrix_name_str
8674     { \@@_node_left:nn #1 { } }
8675     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8676     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8677     \pgftransformshift
8678     {
8679         \pgfpoint
8680         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8681         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8682     }
8683     \str_if_empty:NTF \l_@@_submatrix_name_str
8684     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8685     {
8686         \@@_node_right:nnnn #2
8687         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8688     }

```

```

8689 \cs_set_eq:NN \pgfpointanchor \@_pgfpointanchor:n
8690 \flag_clear_new:N \l_@@_code_flag
8691 \l_@@_code_tl
8692 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $row-i$, $col-j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8693 \cs_set_eq:NN \@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

8694 \cs_new_protected:Npn \@_pgfpointanchor:n #1
8695 {
8696   \use:e
8697   { \exp_not:N \@_old_pgfpointanchor { \@_pgfpointanchor_i:nn #1 } }
8698 }

```

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 -.

```

8699 \cs_new:Npn \@_pgfpointanchor_i:nn #1 #2
8700 { #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.

```

8701 \tl_const:Nn \c_@@_integers_alist_tl
8702 {
8703   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8704   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8705   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8706   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8707 }

```

```

8708 \cs_new:Npn \@_pgfpointanchor_ii:w #1-#2\q_stop
8709 {

```

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`.

```

8710 \tl_if_empty:nTF { #2 }
8711 {
8712   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8713   {
8714     \flag_raise:N \l_@@_code_flag
8715     \int_if_even:nTF { \flag_height:N \l_@@_code_if_flag }
8716     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8717     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8718   }
8719   { #1 }
8720 }

```


If there is an hyphen, we have to see whether we have a node of the form $i-j$, $row-i$ or $col-j$.

```
8721     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8722   }
```

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`).

```
8723 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8724   {
8725     \str_case:nnF { #1 }
8726     {
8727       { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8728       { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8729     }
```

Now the case of a node of the form $i-j$.

```
8730     {
8731       \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8732       - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8733     }
8734   }
```

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`).

```
8735 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8736   {
8737     \pgfnode
8738     { rectangle }
8739     { east }
8740     {
8741       \nullfont
8742       \c_math_toggle_token
8743       \@@_color:o \l_@@_delimiters_color_tl
8744       \left #1
8745       \vcenter
8746       {
8747         \nullfont
8748         \hrule \@height \l_tmpa_dim
8749         \@depth \c_zero_dim
8750         \@width \c_zero_dim
8751       }
8752       \right .
8753       \c_math_toggle_token
8754     }
8755     { #2 }
8756     { }
8757   }
```

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.

```
8758 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8759   {
8760     \pgfnode
8761     { rectangle }
8762     { west }
8763     {
8764       \nullfont
8765       \c_math_toggle_token
8766       \colorlet { current-color } { . }
8767       \@@_color:o \l_@@_delimiters_color_tl
8768       \left .
```

```

8769     \vcenter
8770     {
8771         \nullfont
8772         \hrule \@height \l_tmpa_dim
8773             \@depth \c_zero_dim
8774             \@width \c_zero_dim
8775     }
8776     \right #1
8777     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8778     ~ { \color { current-color } \smash { #4 } }
8779     \c_math_toggle_token
8780 }
8781 { #2 }
8782 { }
8783 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8784 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8785 {
8786     \peek_remove_spaces:n
8787     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8788 }
8789 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8790 {
8791     \peek_remove_spaces:n
8792     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8793 }
8794 \keys_define:nn { nicematrix / Brace }
8795 {
8796     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8797     left-shorten .default:n = true ,
8798     left-shorten .value_forbidden:n = true ,
8799     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8800     right-shorten .default:n = true ,
8801     right-shorten .value_forbidden:n = true ,
8802     shorten .meta:n = { left-shorten , right-shorten } ,
8803     shorten .value_forbidden:n = true ,
8804     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8805     yshift .value_required:n = true ,
8806     yshift .initial:n = \c_zero_dim ,
8807     color .tl_set:N = \l_tmpa_tl ,
8808     color .value_required:n = true ,
8809     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8810 }

```

#1 is the first cell of the rectangle (with the syntax $i-lj$; **#2** is the last cell of the rectangle; **#3** is the label of the text; **#4** is the optional argument (a list of *key-value* pairs); **#5** is equal to `under` or `over`.

```

8811 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8812 {
8813     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8814 \@@_compute_i_j:nn { #1 } { #2 }
8815 \bool_lazy_or:nnTF

```

```

8816 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8817 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8818 {
8819   \str_if_eq:eeTF { #5 } { under }
8820   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8821   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8822 }
8823 {
8824   \tl_clear:N \l_tmpa_tl
8825   \keys_set:nn { nicematrix / Brace } { #4 }
8826   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8827   \pgfpicture
8828   \pgfrememberpicturepositiononpagetrue
8829   \pgf@relevantforpicturesizefalse
8830   \bool_if:NT \l_@@_brace_left_shorten_bool
8831   {
8832     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8833     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8834     {
8835       \cs_if_exist:cT
8836       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8837       {
8838         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8839
8840         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8841         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8842       }
8843     }
8844   }
8845   \bool_lazy_or:nnT
8846   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8847   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8848   {
8849     \@@_qpoint:n { col - \l_@@_first_j_tl }
8850     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8851   }
8852   \bool_if:NT \l_@@_brace_right_shorten_bool
8853   {
8854     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8855     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8856     {
8857       \cs_if_exist:cT
8858       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8859       {
8860         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8861         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8862         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8863       }
8864     }
8865   }
8866   \bool_lazy_or:nnT
8867   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8868   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8869   {
8870     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8871     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8872   }
8873   \pgfset { inner~sep = \c_zero_dim }
8874   \str_if_eq:eeTF { #5 } { under }
8875   { \@@_underbrace_i:n { #3 } }
8876   { \@@_overbrace_i:n { #3 } }
8877   \endpgfpicture
8878 }

```

```

8879   \group_end:
8880 }

```

The argument is the text to put above the brace.

```

8881 \cs_new_protected:Npn \@@_overbrace_i:n #1
8882 {
8883   \@@_qpoint:n { row - \l_@@_first_i_tl }
8884   \pgftransformshift
8885   {
8886     \pgfpoint
8887     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8888     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8889   }
8890   \pgfnode
8891   { rectangle }
8892   { south }
8893   {
8894     \vtop
8895     {
8896       \group_begin:
8897       \everycr { }
8898       \halign
8899       {
8900         \hfil ## \hfil \crcr
8901         \bool_if:NTF \l_@@_tabular_bool
8902         { \begin { tabular } { c } #1 \end { tabular } }
8903         { $ \begin { array } { c } #1 \end { array } $ }
8904         \cr
8905         \c_math_toggle_token
8906         \overbrace
8907         {
8908           \hbox_to_wd:nn
8909           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8910           { }
8911         }
8912         \c_math_toggle_token
8913         \cr
8914       }
8915       \group_end:
8916     }
8917   }
8918   { }
8919   { }
8920 }

```

The argument is the text to put under the brace.

```

8921 \cs_new_protected:Npn \@@_underbrace_i:n #1
8922 {
8923   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8924   \pgftransformshift
8925   {
8926     \pgfpoint
8927     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8928     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8929   }
8930   \pgfnode
8931   { rectangle }
8932   { north }
8933   {
8934     \group_begin:
8935     \everycr { }
8936     \vbox
8937     {

```

```

8938     \halign
8939     {
8940     \hfil ## \hfil \crcr
8941     \c_math_toggle_token
8942     \underbrace
8943     {
8944     \hbox_to_wd:nn
8945     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8946     { }
8947     }
8948     \c_math_toggle_token
8949     \cr
8950     \bool_if:NTF \l_@@_tabular_bool
8951     { \begin { tabular } { c } #1 \end { tabular } }
8952     { $ \begin { array } { c } #1 \end { array } $ }
8953     \cr
8954     }
8955   }
8956   \group_end:
8957 }
8958 { }
8959 { }
8960 }

```

35 The command TikzEveryCell

```

8961 \bool_new:N \l_@@_not_empty_bool
8962 \bool_new:N \l_@@_empty_bool
8963
8964 \keys_define:nn { nicematrix / TikzEveryCell }
8965 {
8966   not-empty .code:n =
8967     \bool_lazy_or:nnTF
8968     \l_@@_in_code_after_bool
8969     \g_@@_recreate_cell_nodes_bool
8970     { \bool_set_true:N \l_@@_not_empty_bool }
8971     { \@@_error:n { detection~of~empty~cells } } } ,
8972   not-empty .value_forbidden:n = true ,
8973   empty .code:n =
8974     \bool_lazy_or:nnTF
8975     \l_@@_in_code_after_bool
8976     \g_@@_recreate_cell_nodes_bool
8977     { \bool_set_true:N \l_@@_empty_bool }
8978     { \@@_error:n { detection~of~empty~cells } } } ,
8979   empty .value_forbidden:n = true ,
8980   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8981 }
8982
8983
8984 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8985 {
8986   \IfPackageLoadedTF { tikz }
8987   {
8988     \group_begin:
8989     \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

8990     \tl_set:Nn \l_tmpa_tl { { #2 } }
8991     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

8992         { \@@_for_a_block:nnnnn #1 }
8993     \@@_all_the_cells:
8994     \group_end:
8995 }
8996 { \@@_error:n { TikzEveryCell~without~tikz } }
8997 }
8998
8999 \tl_new:N \@@_i_tl
9000 \tl_new:N \@@_j_tl
9001
9002
9003 \cs_new_protected:Nn \@@_all_the_cells:
9004 {
9005     \int_step_variable:nNn \c@iRow \@@_i_tl
9006     {
9007         \int_step_variable:nNn \c@jCol \@@_j_tl
9008         {
9009             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9010             {
9011                 \clist_if_in:NcF \l_@@_corners_cells_clist
9012                 { \@@_i_tl - \@@_j_tl }
9013                 {
9014                     \bool_set_false:N \l_tmpa_bool
9015                     \cs_if_exist:cTF
9016                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9017                     {
9018                         \bool_if:NF \l_@@_empty_bool
9019                         { \bool_set_true:N \l_tmpa_bool }
9020                     }
9021                     {
9022                         \bool_if:NF \l_@@_not_empty_bool
9023                         { \bool_set_true:N \l_tmpa_bool }
9024                     }
9025                     \bool_if:NT \l_tmpa_bool
9026                     {
9027                         \@@_block_tikz:onnnn
9028                         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9029                     }
9030                 }
9031             }
9032         }
9033     }
9034 }
9035
9036 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9037 {
9038     \bool_if:NF \l_@@_empty_bool
9039     {
9040         \@@_block_tikz:onnnn
9041         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9042     }
9043     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9044 }
9045
9046 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9047 {
9048     \int_step_inline:nnn { #1 } { #3 }
9049     {
9050         \int_step_inline:nnn { #2 } { #4 }
9051         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9052     }
9053 }

```

36 The command \ShowCellNames

```

9054 \NewDocumentCommand \@@_ShowCellNames { }
9055 {
9056   \bool_if:NT \l_@@_in_code_after_bool
9057   {
9058     \pgfpicture
9059     \pgfrememberpicturepositiononpagetrue
9060     \pgf@relevantforpicturesizefalse
9061     \pgfpathrectanglecorners
9062     { \@@_qpoint:n { 1 } }
9063     {
9064       \@@_qpoint:n
9065       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9066     }
9067     \pgfsetfillopacity { 0.75 }
9068     \pgfsetfillcolor { white }
9069     \pgfusepathqfill
9070     \endpgfpicture
9071   }
9072   \dim_gzero_new:N \g_@@_tmpc_dim
9073   \dim_gzero_new:N \g_@@_tmpd_dim
9074   \dim_gzero_new:N \g_@@_tmpe_dim
9075   \int_step_inline:nn \c@iRow
9076   {
9077     \bool_if:NTF \l_@@_in_code_after_bool
9078     {
9079       \pgfpicture
9080       \pgfrememberpicturepositiononpagetrue
9081       \pgf@relevantforpicturesizefalse
9082     }
9083     { \begin { pgfpicture } }
9084     \@@_qpoint:n { row - ##1 }
9085     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9086     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9087     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9088     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9089     \bool_if:NTF \l_@@_in_code_after_bool
9090     { \endpgfpicture }
9091     { \end { pgfpicture } }
9092     \int_step_inline:nn \c@jCol
9093     {
9094       \hbox_set:Nn \l_tmpa_box
9095       {
9096         \normalfont \Large \sffamily \bfseries
9097         \bool_if:NTF \l_@@_in_code_after_bool
9098         { \color { red } }
9099         { \color { red ! 50 } }
9100         ##1 - #####1
9101       }
9102       \bool_if:NTF \l_@@_in_code_after_bool
9103       {
9104         \pgfpicture
9105         \pgfrememberpicturepositiononpagetrue
9106         \pgf@relevantforpicturesizefalse
9107       }
9108       { \begin { pgfpicture } }
9109       \@@_qpoint:n { col - #####1 }
9110       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9111       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9112       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9113       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9114     \bool_if:NTF \l_@@_in_code_after_bool
9115     { \endpgfpicture }
9116     { \end { pgfpicture } }
9117     \fp_set:Nn \l_tmpa_fp
9118     {
9119         \fp_min:nn
9120         {
9121             \fp_min:nn
9122             { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9123             { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9124         }
9125         { 1.0 }
9126     }
9127     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9128     \pgfpicture
9129     \pgfrememberpicturerepositiononpagetrue
9130     \pgf@relevantforpicturesizefalse
9131     \pgftransformshift
9132     {
9133         \pgfpoint
9134         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9135         { \dim_use:N \g_tmpa_dim }
9136     }
9137     \pgfnode
9138     { rectangle }
9139     { center }
9140     { \box_use:N \l_tmpa_box }
9141     { }
9142     { }
9143     \endpgfpicture
9144 }
9145 }
9146 }

```

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.

```

9147 \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.

```

9148 \bool_new:N \g_@@_footnote_bool
9149 \msg_new:nmmm { nicematrix } { Unknown~key~for~package }
9150 {
9151     The~key~'\l_keys_key_str'~is~unknown. \\
9152     That~key~will~be~ignored. \\
9153     For~a~list~of~the~available~keys,~type~H~<return>.
9154 }
9155 {
9156     The~available~keys~are~(in~alphabetic~order):~
9157     footnote,~
9158     footnotehyper,~
9159     messages~for~Overleaf,~
9160     renew~dots,~and~
9161     renew~matrix.

```



```

9162 }
9163 \keys_define:nn { nicematrix / Package }
9164 {
9165   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9166   renew-dots .value_forbidden:n = true ,
9167   renew-matrix .code:n = \@@_renew_matrix: ,
9168   renew-matrix .value_forbidden:n = true ,
9169   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9170   footnote .bool_set:N = \g_@@_footnote_bool ,
9171   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. You keep the following key only for compatibility but maybe we will delete it.

```

9172   no-test-for-array .code:n = \prg_do_nothing: ,
9173   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9174 }
9175 \ProcessKeysOptions { nicematrix / Package }

```

```

9176 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9177 {
9178   You-can't-use-the-option-'footnote'~because~the~package~
9179   footnotehyper~has~already~been~loaded.~
9180   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9181   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9182   of~the~package~footnotehyper.\\
9183   The~package~footnote~won't~be~loaded.
9184 }

```

```

9185 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9186 {
9187   You-can't-use-the-option-'footnotehyper'~because~the~package~
9188   footnote~has~already~been~loaded.~
9189   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9190   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9191   of~the~package~footnote.\\
9192   The~package~footnotehyper~won't~be~loaded.
9193 }

```

```

9194 \bool_if:NT \g_@@_footnote_bool
9195 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9196   \IfClassLoadedTF { beamer }
9197     { \bool_set_false:N \g_@@_footnote_bool }
9198     {
9199       \IfPackageLoadedTF { footnotehyper }
9200         { \@@_error:n { footnote~with~footnotehyper~package } }
9201         { \usepackage { footnote } }
9202     }
9203 }

```

```

9204 \bool_if:NT \g_@@_footnotehyper_bool
9205 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9206   \IfClassLoadedTF { beamer }
9207     { \bool_set_false:N \g_@@_footnote_bool }
9208     {
9209       \IfPackageLoadedTF { footnote }
9210         { \@@_error:n { footnotehyper~with~footnote~package } }
9211         { \usepackage { footnotehyper } }
9212     }

```

```

9213   \bool_set_true:N \g_@@_footnote_bool
9214 }

```

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.

```

9215 \bool_new:N \l_@@_underscore_loaded_bool
9216 \IfPackageLoadedT { underscore }
9217 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9218 \hook_gput_code:nnn { begindocument } { . }
9219 {
9220   \bool_if:NF \l_@@_underscore_loaded_bool
9221   {
9222     \IfPackageLoadedT { underscore }
9223     { \@@_error:n { underscore-after~nicematrix } }
9224   }
9225 }

```

39 Error messages of the package

```

9226 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9227 { \str_const:Nn \c_@@_available_keys_str { } }
9228 {
9229   \str_const:Nn \c_@@_available_keys_str
9230   { For~a~list~of~the~available~keys,~type-H~<return>. }
9231 }
9232 \seq_new:N \g_@@_types_of_matrix_seq
9233 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9234 {
9235   NiceMatrix ,
9236   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9237 }
9238 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9239 { \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`.

```

9240 \cs_new_protected:Npn \@@_error_too_much_cols:
9241 {
9242   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9243   { \@@_fatal:nm { too-much-cols-for-array } }
9244   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9245   { \@@_fatal:n { too-much-cols-for-matrix } }
9246   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9247   { \@@_fatal:n { too-much-cols-for-matrix } }
9248   \bool_if:NF \l_@@_last_col_without_value_bool
9249   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }

```

9250 }

The following command must *not* be protected since it's used in an error message.

```
9251 \cs_new:Npn \@@_message_hdotsfor:
9252 {
9253   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9254   { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
9255 }
9256 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9257 {
9258   Incompatible-options.\\
9259   You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-this-time.\\
9260   The-output-will-not-be-reliable.
9261 }
9262 \@@_msg_new:nn { negative-weight }
9263 {
9264   Negative-weight.\\
9265   The-weight-of-the-'X'-columns-must-be-positive-and-you-have-used-
9266   the-value-\int_use:N \l_@@_weight_int'.\\
9267   The-absolute-value-will-be-used.
9268 }
9269 \@@_msg_new:nn { last-col-not-used }
9270 {
9271   Column-not-used.\\
9272   The-key-'last-col'-is-in-force-but-you-have-not-used-that-last-column-
9273   in-your-\@@_full_name_env:.~However,~you-can-go-on.
9274 }
9275 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9276 {
9277   Too-much-columns.\\
9278   In-the-row~\int_eval:n { \c@iRow },~
9279   you-try-to-use-more-columns-
9280   than-allowed-by-your-\@@_full_name_env:. \@@_message_hdotsfor:\
9281   The-maximal-number-of-columns-is-\int_eval:n { \l_@@_last_col_int - 1 }~
9282   (plus-the-exterior-columns).~This-error-is-fatal.
9283 }
9284 \@@_msg_new:nn { too-much-cols-for-matrix }
9285 {
9286   Too-much-columns.\\
9287   In-the-row~\int_eval:n { \c@iRow },~
9288   you-try-to-use-more-columns-than-allowed-by-your~
9289   \@@_full_name_env:. \@@_message_hdotsfor:\ Recall-that-the-maximal-
9290   number-of-columns-for-a-matrix~(excepted-the-potential-exterior-
9291   columns)-is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
9292   Its-current-value-is-\int_use:N \c@MaxMatrixCols\ (use~
9293   \token_to_str:N \setcounter\ to-change-that-value).~
9294   This-error-is-fatal.
9295 }
9296 \@@_msg_new:nn { too-much-cols-for-array }
9297 {
9298   Too-much-columns.\\
9299   In-the-row~\int_eval:n { \c@iRow },~
9300   ~you-try-to-use-more-columns-than-allowed-by-your~
9301   \@@_full_name_env:. \@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
9302   \int_use:N \g_@@_static_num_of_col_int\
9303   ~ (plus-the-potential-exterior-ones).~
9304   This-error-is-fatal.
9305 }
9306 \@@_msg_new:nn { columns-not-used }
9307 {
9308   Columns-not-used.\\
```

```

9309     The-preamble-of-your-@@_full_name_env:\ announces-\int_use:N
9310     \g_@@_static_num_of_col_int\ columns-but-you-use-only-\int_use:N \c@jCol.\\
9311     The-columns-you-did-not-used-won't-be-created.\\
9312     You-won't-have-similar-error-message-till-the-end-of-the-document.
9313 }
9314 \@@_msg_new:nn { empty-preamble }
9315 {
9316     Empty-preamble.\\
9317     The-preamble-of-your-@@_full_name_env:\ is-empty.\\
9318     This-error-is-fatal.
9319 }
9320 \@@_msg_new:nn { in-first-col }
9321 {
9322     Erroneous-use.\\
9323     You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9324     That-command-will-be-ignored.
9325 }
9326 \@@_msg_new:nn { in-last-col }
9327 {
9328     Erroneous-use.\\
9329     You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9330     That-command-will-be-ignored.
9331 }
9332 \@@_msg_new:nn { in-first-row }
9333 {
9334     Erroneous-use.\\
9335     You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9336     That-command-will-be-ignored.
9337 }
9338 \@@_msg_new:nn { in-last-row }
9339 {
9340     You-can't-use-the-command-#1 in-the-last-row-(exterior)-of-the-array.\\
9341     That-command-will-be-ignored.
9342 }
9343 \@@_msg_new:nn { caption-outside-float }
9344 {
9345     Key-caption-forbidden.\\
9346     You-can't-use-the-key-'caption'-because-you-are-not-in-a-floating-
9347     environment.-This-key-will-be-ignored.
9348 }
9349 \@@_msg_new:nn { short-caption-without-caption }
9350 {
9351     You-should-not-use-the-key-'short-caption'-without-'caption'.-
9352     However,-your-'short-caption'-will-be-used-as-'caption'.
9353 }
9354 \@@_msg_new:nn { double-closing-delimiter }
9355 {
9356     Double-delimiter.\\
9357     You-can't-put-a-second-closing-delimiter-"#1"-just-after-a-first-closing-
9358     delimiter.-This-delimiter-will-be-ignored.
9359 }
9360 \@@_msg_new:nn { delimiter-after-opening }
9361 {
9362     Double-delimiter.\\
9363     You-can't-put-a-second-delimiter-"#1"-just-after-a-first-opening-
9364     delimiter.-That-delimiter-will-be-ignored.
9365 }
9366 \@@_msg_new:nn { bad-option-for-line-style }
9367 {

```

```

9368     Bad-line-style.\
9369     Since-you-haven't-loaded-Tikz,~the-only~value~you~can~give~to~'line-style'~
9370     is~'standard'.~That~key~will~be~ignored.
9371 }
9372 \@@_msg_new:nn { Identical-notes-in-caption }
9373 {
9374     Identical~tabular~notes.\
9375     You~can't~put~several~notes~with~the~same~content~in~
9376     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\
9377     If~you~go~on,~the~output~will~probably~be~erroneous.
9378 }
9379 \@@_msg_new:nn { tabularnote~below~the~tabular }
9380 {
9381     \token_to_str:N \tabularnote\ forbidden\
9382     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9383     of~your~tabular~because~the~caption~will~be~composed~below~
9384     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9385     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\
9386     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9387     no~similar~error~will~raised~in~this~document.
9388 }
9389 \@@_msg_new:nn { Unknown~key~for~rules }
9390 {
9391     Unknown~key.\
9392     There~is~only~two~keys~available~here:~width~and~color.\
9393     Your~key~'\l_keys_key_str'~will~be~ignored.
9394 }
9395 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9396 {
9397     Unknown~key.\
9398     There~is~only~two~keys~available~here:~
9399     'empty'~and~'not-empty'.\
9400     Your~key~'\l_keys_key_str'~will~be~ignored.
9401 }
9402 \@@_msg_new:nn { Unknown~key~for~rotate }
9403 {
9404     Unknown~key.\
9405     The~only~key~available~here~is~'c'.\
9406     Your~key~'\l_keys_key_str'~will~be~ignored.
9407 }
9408 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9409 {
9410     Unknown~key.\
9411     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9412     It~you~go~on,~you~will~probably~have~other~errors. \
9413     \c_@@_available_keys_str
9414 }
9415 {
9416     The~available~keys~are~(in~alphabetic~order):~
9417     ccommand,~
9418     color,~
9419     command,~
9420     dotted,~
9421     letter,~
9422     multiplicity,~
9423     sep-color,~
9424     tikz,~and~total-width.
9425 }
9426 \@@_msg_new:nnn { Unknown~key~for~xdots }
9427 {
9428     Unknown~key.\

```

```

9429     The-key~'\l_keys_key_str'~is-unknown~for~a~command~for~drawing~dotted~rules.\\
9430     \c_@@_available_keys_str
9431   }
9432   {
9433     The-available-keys-are-(in-alphabetic-order):~
9434     'color',~
9435     'horizontal-labels',~
9436     'inter',~
9437     'line-style',~
9438     'radius',~
9439     'shorten',~
9440     'shorten-end'~and~'shorten-start'.
9441   }

9442 \@@_msg_new:nn { Unknown-key~for~rowcolors }
9443   {
9444     Unknown-key.\\
9445     As~for~now,~there-is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9446     (and-you-try-to-use~'\l_keys_key_str')\\
9447     That-key-will-be-ignored.
9448   }

9449 \@@_msg_new:nn { label-without-caption }
9450   {
9451     You-can't-use-the-key~'label'~in-your~'{NiceTabular}'~because~
9452     you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9453   }

9454 \@@_msg_new:nn { W-warning }
9455   {
9456     Line~\msg_line_number:.~The-cell-is-too-wide~for~your~column~'W'~
9457     (row~\int_use:N \c@iRow).
9458   }

9459 \@@_msg_new:nn { Construct-too-large }
9460   {
9461     Construct-too-large.\\
9462     Your~command~\token_to_str:N #1
9463     can't-be-drawn~because~your~matrix~is~too~small.\\
9464     That~command~will~be~ignored.
9465   }

9466 \@@_msg_new:nn { underscore-after-nicematrix }
9467   {
9468     Problem-with~'underscore'.\\
9469     The~package~'underscore'~should-be-loaded-before~'nicematrix'.~
9470     You-can-go-on-but-you-won't-be-able-to-write~something~such~as:\\
9471     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9472   }

9473 \@@_msg_new:nn { ampersand-in-light-syntax }
9474   {
9475     Ampersand-forbidden.\\
9476     You-can't-use-an-ampersand~(\token_to_str:N &)~to~separate~columns~because~
9477     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9478   }

9479 \@@_msg_new:nn { double-backslash-in-light-syntax }
9480   {
9481     Double~backslash~forbidden.\\
9482     You-can't-use~\token_to_str:N
9483     \\~to~separate~rows~because~the~key~'light-syntax'~
9484     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9485     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9486   }

9487 \@@_msg_new:nn { hlines-with-color }
9488   {

```

```

9489     Incompatible-keys.\
9490     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for~a~
9491     '\token_to_str:N \Block'~when~the-key~'color'~or~'draw'~is~used.\
9492     However,~you-can~put~several~commands~\token_to_str:N \Block.\
9493     Your-key-will-be-discarded.
9494 }

9495 \@@_msg_new:nn { bad-value-for-baseline }
9496 {
9497     Bad-value-for-baseline.\
9498     The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9499     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9500     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9501     the-form-'line-i'.\
9502     A-value-of~1~will-be-used.
9503 }

9504 \@@_msg_new:nn { detection-of-empty-cells }
9505 {
9506     Problem-with-'not-empty'\
9507     For-technical-reasons,~you-must-activate~
9508     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9509     in-order-to-use-the-key~'\l_keys_key_str'.\
9510     That-key-will-be-ignored.
9511 }

9512 \@@_msg_new:nn { siunitx-not-loaded }
9513 {
9514     siunitx-not-loaded\
9515     You-can't-use-the-columns-'S'~because-'siunitx'~is~not~loaded.\
9516     That-error-is-fatal.
9517 }

9518 \@@_msg_new:nn { Invalid-name }
9519 {
9520     Invalid-name.\
9521     You-can't-give-the-name~'\l_keys_value_tl'~to~a~\token_to_str:N
9522     \SubMatrix\ of~your~\@@_full_name_env:.\
9523     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\
9524     This-key-will-be-ignored.
9525 }

9526 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9527 {
9528     Wrong-line.\
9529     You-try-to-draw~a~#1~line-of-number~'#2'~in~a~
9530     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9531     number~is~not~valid.~It~will-be-ignored.
9532 }

9533 \@@_msg_new:nn { Impossible-delimiter }
9534 {
9535     Impossible-delimiter.\
9536     It's-impossible-to-draw-the-#1-delimiter-of~your~
9537     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9538     in~that~column.
9539     \bool_if:NT \l_@@_submatrix_slim_bool
9540     { ~Maybe-you-should-try-without-the-key-'slim'. } \
9541     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9542 }

9543 \@@_msg_new:nnn { width-without-X-columns }
9544 {
9545     You-have-used-the-key~'width'~but~you-have-put~no~'X'~column.~
9546     That-key-will-be-ignored.
9547 }
9548 {
9549     This-message-is~the-message-'width-without-X-columns'~

```

```

9550   of~the~module~'nicematrix'.~
9551   The~experimented~users~can~disable~that~message~with~
9552   \token_to_str:N \msg_redirect_name:nnn.\\
9553 }
9554
9555 \@@_msg_new:nn { key~multiplicity~with~dotted }
9556 {
9557   Incompatible~keys. \\
9558   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9559   in~a~'custom~line'.~They~are~incompatible. \\
9560   The~key~'multiplicity'~will~be~discarded.
9561 }
9562 \@@_msg_new:nn { empty~environment }
9563 {
9564   Empty~environment.\\
9565   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9566 }
9567 \@@_msg_new:nn { No~letter~and~no~command }
9568 {
9569   Erroneous~use.\\
9570   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
9571   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9572   '~'command'~(to~draw~horizontal~rules).\\
9573   However,~you~can~go~on.
9574 }
9575 \@@_msg_new:nn { Forbidden~letter }
9576 {
9577   Forbidden~letter.\\
9578   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9579   It~will~be~ignored.
9580 }
9581 \@@_msg_new:nn { Several~letters }
9582 {
9583   Wrong~name.\\
9584   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9585   have~used~'\l_@@_letter_str').\\
9586   It~will~be~ignored.
9587 }
9588 \@@_msg_new:nn { Delimiter~with~small }
9589 {
9590   Delimiter~forbidden.\\
9591   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9592   because~the~key~'small'~is~in~force.\\
9593   This~error~is~fatal.
9594 }
9595 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9596 {
9597   Unknown~cell.\\
9598   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9599   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9600   can't~be~executed~because~a~cell~doesn't~exist.\\
9601   This~command~\token_to_str:N \line\ will~be~ignored.
9602 }
9603 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9604 {
9605   Duplicate~name.\\
9606   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9607   in~this~\@@_full_name_env:.\\
9608   This~key~will~be~ignored.\\
9609   \bool_if:NF \g_@@_messages_for_Overleaf_bool

```



```

9610     { For-a-list-of-the-names-already-used,-type-H<return>. }
9611   }
9612   {
9613     The-names-already-defined-in-this-\@@_full_name_env:\ are:~
9614     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9615   }
9616 \@@_msg_new:nn { r~or~l~with~preamble }
9617   {
9618     Erroneous-use.\\
9619     You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~.~
9620     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of~
9621     your~\@@_full_name_env:~.\\
9622     This-key-will-be-ignored.
9623   }
9624 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9625   {
9626     Erroneous-use.\\
9627     You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9628     the~array.~This~error~is~fatal.
9629   }
9630 \@@_msg_new:nn { bad~corner }
9631   {
9632     Bad-corner.\\
9633     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9634     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9635     This~specification~of~corner~will~be~ignored.
9636   }
9637 \@@_msg_new:nn { bad~border }
9638   {
9639     Bad-border.\\
9640     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9641     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9642     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9643     also~use~the~key~'tikz'
9644     \IfPackageLoadedF { tikz }
9645     {-if-you-load~the~LaTeX~package~'tikz'}).\\
9646     This~specification~of~border~will~be~ignored.
9647   }
9648 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9649   {
9650     TikZ~not~loaded.\\
9651     You-can't-use~\token_to_str:N \TikzEveryCell\
9652     because~you~have~not~loaded~tikz.~
9653     This~command~will~be~ignored.
9654   }
9655 \@@_msg_new:nn { tikz~key~without~tikz }
9656   {
9657     TikZ~not~loaded.\\
9658     You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
9659     \Block'~because~you~have~not~loaded~tikz.~
9660     This~key~will~be~ignored.
9661   }
9662 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9663   {
9664     Erroneous-use.\\
9665     In~the~\@@_full_name_env:,~you~must~use~the~key~
9666     'last~col'~without~value.\\
9667     However,~you~can~go~on~for~this~time~
9668     (the~value~'\l_keys_value_tl'~will~be~ignored).
9669   }

```

```

9670 \@@_msg_new:nn { last-col-non-empty-for-NiceMatrixOptions }
9671 {
9672   Erroneous-use.\
9673   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9674   'last-col'~without-value.\
9675   However,~you-can-go-on-for-this-time~
9676   (the-value~'\l_keys_value_tl'~will-be-ignored).
9677 }

9678 \@@_msg_new:nn { Block-too-large-1 }
9679 {
9680   Block-too-large.\
9681   You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
9682   too-small-for-that-block. \
9683   This-block-and-maybe-others-will-be-ignored.
9684 }

9685 \@@_msg_new:nn { Block-too-large-2 }
9686 {
9687   Block-too-large.\
9688   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9689   \g_@@_static_num_of_col_int\
9690   columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9691   specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9692   (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9693   This-block-and-maybe-others-will-be-ignored.
9694 }

9695 \@@_msg_new:nn { unknown-column-type }
9696 {
9697   Bad-column-type.\
9698   The-column-type~'#1'~in-your~\@@_full_name_env:\
9699   is-unknown. \
9700   This-error-is-fatal.
9701 }

9702 \@@_msg_new:nn { unknown-column-type-S }
9703 {
9704   Bad-column-type.\
9705   The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \
9706   If-you-want-to-use-the-column-type~'S'~of~siunitx,~you-should~
9707   load-that-package. \
9708   This-error-is-fatal.
9709 }

9710 \@@_msg_new:nn { tabularnote-forbidden }
9711 {
9712   Forbidden-command.\
9713   You-can't-use-the-command~\token_to_str:N\tabularnote\
9714   ~here.~This-command-is-available-only-in~
9715   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9716   the-argument-of-a-command~\token_to_str:N \caption\ included~
9717   in-an-environment~{table}. \
9718   This-command-will-be-ignored.
9719 }

9720 \@@_msg_new:nn { borders-forbidden }
9721 {
9722   Forbidden-key.\
9723   You-can't-use-the-key~'borders'~of~the-command~\token_to_str:N \Block\
9724   because-the-option~'rounded-corners'~
9725   is-in-force-with-a-non-zero-value.\
9726   This-key-will-be-ignored.
9727 }

9728 \@@_msg_new:nn { bottomrule-without-booktabs }
9729 {
9730   booktabs-not-loaded.\

```

```

9731     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9732     loaded~'booktabs'.\\
9733     This~key~will~be~ignored.
9734 }

9735 \@@_msg_new:nn { enumitem~not~loaded }
9736 {
9737     enumitem~not~loaded.\\
9738     You~can't~use~the~command~\token_to_str:N\tabularnote\
9739     ~because~you~haven't~loaded~'enumitem'.\\
9740     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9741     ignored~in~the~document.
9742 }

9743 \@@_msg_new:nn { tikz~without~tikz }
9744 {
9745     Tikz~not~loaded.\\
9746     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9747     loaded.~If~you~go~on,~that~key~will~be~ignored.
9748 }

9749 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9750 {
9751     Tikz~not~loaded.\\
9752     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9753     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9754     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9755     use~that~custom~line.
9756 }

9757 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9758 {
9759     Tikz~not~loaded.\\
9760     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9761     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9762     That~key~will~be~ignored.
9763 }

9764 \@@_msg_new:nn { without~color~inside }
9765 {
9766     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9767     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9768     outside~\token_to_str:N \CodeBefore,~you~
9769     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9770     You~can~go~on~but~you~may~need~more~compilations.
9771 }

9772 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9773 {
9774     Erroneous~use.\\
9775     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9776     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9777     The~key~'color'~will~be~discarded.
9778 }

9779 \@@_msg_new:nn { Wrong~last~row }
9780 {
9781     Wrong~number.\\
9782     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9783     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9784     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9785     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9786     without~value~(more~compilations~might~be~necessary).
9787 }

9788 \@@_msg_new:nn { Yet~in~env }
9789 {
9790     Nested~environments.\\

```

```

9791     Environments-of-nicematrix-can't-be-nested.\
9792     This-error-is-fatal.
9793 }
9794 \@@_msg_new:nn { Outside-math-mode }
9795 {
9796     Outside-math-mode.\
9797     The-\@@_full_name_env:\ can-be-used-only-in-math-mode~
9798     (and-not-in-\token_to_str:N \vcenter).\
9799     This-error-is-fatal.
9800 }
9801 \@@_msg_new:nn { One-letter-allowed }
9802 {
9803     Bad-name.\
9804     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\
9805     It-will-be-ignored.
9806 }
9807 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9808 {
9809     Environment~{TabularNote}~forbidden.\
9810     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9811     but~*before*~the~\token_to_str:N \CodeAfter.\
9812     This-environment~{TabularNote}~will-be-ignored.
9813 }
9814 \@@_msg_new:nn { varwidth-not-loaded }
9815 {
9816     varwidth-not-loaded.\
9817     You-can't-use-the-column-type~'V'~because~'varwidth'~is-not~
9818     loaded.\
9819     Your-column-will-behave-like~'p'.
9820 }
9821 \@@_msg_new:nnn { Unknow-key-for-RulesBis }
9822 {
9823     Unknow-key.\
9824     Your-key~'\l_keys_key_str'~is-unknown-for-a-rule.\
9825     \c_@@_available_keys_str
9826 }
9827 {
9828     The-available-keys-are-(in-alphabetic-order):~
9829     color,~
9830     dotted,~
9831     multiplicity,~
9832     sep-color,~
9833     tikz,~and~total-width.
9834 }
9835
9836 \@@_msg_new:nnn { Unknow-key-for-Block }
9837 {
9838     Unknow-key.\
9839     The-key~'\l_keys_key_str'~is-unknown-for-the-command~\token_to_str:N
9840     \Block.\ It-will-be-ignored. \
9841     \c_@@_available_keys_str
9842 }
9843 {
9844     The-available-keys-are-(in-alphabetic-order):~&-in-blocks,~ampersand-in-blocks,~
9845     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9846     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9847     and~vlines.
9848 }
9849 \@@_msg_new:nnn { Unknow-key-for-Brace }
9850 {
9851     Unknow-key.\

```

```

9852 The-key~'\l_keys_key_str'~is-unknown~for~the~commands~\token_to_str:N
9853 \UnderBrace\ and~\token_to_str:N \OverBrace.\
9854 It~will~be~ignored. \
9855 \c_@@_available_keys_str
9856 }
9857 {
9858 The-available-keys-are~(in~alphabetic~order):~color,~left-shorten,~
9859 right-shorten,~shorten~(which~fixes~both~left~shorten~and~
9860 right-shorten)~and~yshift.
9861 }
9862 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9863 {
9864 Unknown~key.\
9865 The-key~'\l_keys_key_str'~is-unknown.\
9866 It~will~be~ignored. \
9867 \c_@@_available_keys_str
9868 }
9869 {
9870 The-available-keys-are~(in~alphabetic~order):~
9871 delimiters/color,~
9872 rules~(with~the~subkeys~'color'~and~'width'),~
9873 sub-matrix~(several~subkeys)~
9874 and~xdots~(several~subkeys).~
9875 The-latter~is~for~the~command~\token_to_str:N \line.
9876 }
9877 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9878 {
9879 Unknown~key.\
9880 The-key~'\l_keys_key_str'~is-unknown.\
9881 It~will~be~ignored. \
9882 \c_@@_available_keys_str
9883 }
9884 {
9885 The-available-keys-are~(in~alphabetic~order):~
9886 create-cell-nodes,~
9887 delimiters/color~and~
9888 sub-matrix~(several~subkeys).
9889 }
9890 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9891 {
9892 Unknown~key.\
9893 The-key~'\l_keys_key_str'~is-unknown.\
9894 That-key-will-be-ignored. \
9895 \c_@@_available_keys_str
9896 }
9897 {
9898 The-available-keys-are~(in~alphabetic~order):~
9899 'delimiters/color',~
9900 'extra-height',~
9901 'hlines',~
9902 'hvlines',~
9903 'left-xshift',~
9904 'name',~
9905 'right-xshift',~
9906 'rules'~(with~the~subkeys~'color'~and~'width'),~
9907 'slim',~
9908 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9909 and~'right-xshift').\
9910 }
9911 \@@_msg_new:nnn { Unknown~key~for~notes }
9912 {
9913 Unknown~key.\

```

```

9914     The-key~'\l_keys_key_str'~is-unknown.\\
9915     That-key-will-be-ignored. \\
9916     \c_@@_available_keys_str
9917 }
9918 {
9919     The-available-keys-are-(in~alphabetic-order):~
9920     bottomrule,~
9921     code-after,~
9922     code-before,~
9923     detect-duplicates,~
9924     enumitem-keys,~
9925     enumitem-keys-para,~
9926     para,~
9927     label-in-list,~
9928     label-in-tabular~and~
9929     style.
9930 }
9931 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9932 {
9933     Unknown~key.\\
9934     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9935     \token_to_str:N \RowStyle. \\
9936     That-key-will-be-ignored. \\
9937     \c_@@_available_keys_str
9938 }
9939 {
9940     The-available-keys-are-(in~alphabetic-order):~
9941     'bold',~
9942     'cell-space-top-limit',~
9943     'cell-space-bottom-limit',~
9944     'cell-space-limits',~
9945     'color',~
9946     'nb-rows'~and~
9947     'rowcolor'.
9948 }
9949 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9950 {
9951     Unknown~key.\\
9952     The-key~'\l_keys_key_str'~is-unknown~for~the~command~
9953     \token_to_str:N \NiceMatrixOptions. \\
9954     That-key-will-be-ignored. \\
9955     \c_@@_available_keys_str
9956 }
9957 {
9958     The-available-keys-are-(in~alphabetic-order):~
9959     &in-blocks,~
9960     allow-duplicate-names,~
9961     ampersand-in-blocks,~
9962     caption-above,~
9963     cell-space-bottom-limit,~
9964     cell-space-limits,~
9965     cell-space-top-limit,~
9966     code-for-first-col,~
9967     code-for-first-row,~
9968     code-for-last-col,~
9969     code-for-last-row,~
9970     corners,~
9971     custom-key,~
9972     create-extra-nodes,~
9973     create-medium-nodes,~
9974     create-large-nodes,~
9975     custom-line,~
9976     delimiters~(several~subkeys),~

```

```

9977   end-of-row,~
9978   first-col,~
9979   first-row,~
9980   hlines,~
9981   hvlines,~
9982   hvlines-except-borders,~
9983   last-col,~
9984   last-row,~
9985   left-margin,~
9986   light-syntax,~
9987   light-syntax-expanded,~
9988   matrix/columns-type,~
9989   no-cell-nodes,~
9990   notes~(several~subkeys),~
9991   nullify-dots,~
9992   pgf-node-code,~
9993   renew-dots,~
9994   renew-matrix,~
9995   respect-arraystretch,~
9996   rounded-corners,~
9997   right-margin,~
9998   rules~(with~the~subkeys~'color'~and~'width'),~
9999   small,~
10000  sub-matrix~(several~subkeys),~
10001  vlines,~
10002  xdots~(several~subkeys).
10003  }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10004 \@@_msg_new:nnn { Unknown-key-for-NiceArray }
10005 {
10006   Unknown-key.\\
10007   The-key~'\l_keys_key_str'~is-unknown-for-the-environment-
10008   \{NiceArray\}. \\
10009   That-key-will-be-ignored. \\
10010   \c_@@_available_keys_str
10011 }
10012 {
10013   The-available-keys-are-(in-alphabetic-order):~
10014   &-in-blocks,~
10015   ampersand-in-blocks,~
10016   b,~
10017   baseline,~
10018   c,~
10019   cell-space-bottom-limit,~
10020   cell-space-limits,~
10021   cell-space-top-limit,~
10022   code-after,~
10023   code-for-first-col,~
10024   code-for-first-row,~
10025   code-for-last-col,~
10026   code-for-last-row,~
10027   color-inside,~
10028   columns-width,~
10029   corners,~
10030   create-extra-nodes,~
10031   create-medium-nodes,~
10032   create-large-nodes,~
10033   extra-left-margin,~
10034   extra-right-margin,~
10035   first-col,~
10036   first-row,~
10037   hlines,~

```

```

10038   hvlines,~
10039   hvlines-except-borders,~
10040   last-col,~
10041   last-row,~
10042   left-margin,~
10043   light-syntax,~
10044   light-syntax-expanded,~
10045   name,~
10046   no-cell-nodes,~
10047   nullify-dots,~
10048   pgf-node-code,~
10049   renew-dots,~
10050   respect-arraystretch,~
10051   right-margin,~
10052   rounded-corners,~
10053   rules~(with~the~subkeys~'color'~and~'width'),~
10054   small,~
10055   t,~
10056   vlines,~
10057   xdots/color,~
10058   xdots/shorten-start,~
10059   xdots/shorten-end,~
10060   xdots/shorten-and~
10061   xdots/line-style.
10062 }

```

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`).

```

10063 \@@_msg_new:nmn { Unknown~key~for~NiceMatrix }
10064 {
10065   Unknown~key.\\
10066   The~key~'\l_keys_key_str'~is~unknown~for~the~
10067   \@@_full_name_env:. \\
10068   That~key~will~be~ignored. \\
10069   \c_@@_available_keys_str
10070 }
10071 {
10072   The~available~keys~are~(in~alphabetic~order):~
10073   &~in~blocks,~
10074   ampersand~in~blocks,~
10075   b,~
10076   baseline,~
10077   c,~
10078   cell-space-bottom-limit,~
10079   cell-space-limits,~
10080   cell-space-top-limit,~
10081   code-after,~
10082   code-for-first-col,~
10083   code-for-first-row,~
10084   code-for-last-col,~
10085   code-for-last-row,~
10086   color-inside,~
10087   columns-type,~
10088   columns-width,~
10089   corners,~
10090   create-extra-nodes,~
10091   create-medium-nodes,~
10092   create-large-nodes,~
10093   extra-left-margin,~
10094   extra-right-margin,~
10095   first-col,~
10096   first-row,~
10097   hlines,~
10098   hvlines,~

```



```

10099   hvlines-except-borders,~
10100   l,~
10101   last-col,~
10102   last-row,~
10103   left-margin,~
10104   light-syntax,~
10105   light-syntax-expanded,~
10106   name,~
10107   no-cell-nodes,~
10108   nullify-dots,~
10109   pgf-node-code,~
10110   r,~
10111   renew-dots,~
10112   respect-arraystretch,~
10113   right-margin,~
10114   rounded-corners,~
10115   rules~(with~the~subkeys~'color'~and~'width'),~
10116   small,~
10117   t,~
10118   vlines,~
10119   xdots/color,~
10120   xdots/shorten-start,~
10121   xdots/shorten-end,~
10122   xdots/shorten-and~
10123   xdots/line-style.
10124 }
10125 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10126 {
10127   Unknown~key.\\
10128   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10129   \{NiceTabular\}. \\
10130   That~key~will~be~ignored. \\
10131   \c_@@_available_keys_str
10132 }
10133 {
10134   The~available~keys~are~(in~alphabetic~order):~
10135   &~in~blocks,~
10136   ampersand~in~blocks,~
10137   b,~
10138   baseline,~
10139   c,~
10140   caption,~
10141   cell-space-bottom-limit,~
10142   cell-space-limits,~
10143   cell-space-top-limit,~
10144   code~after,~
10145   code~for~first~col,~
10146   code~for~first~row,~
10147   code~for~last~col,~
10148   code~for~last~row,~
10149   color~inside,~
10150   columns~width,~
10151   corners,~
10152   custom~line,~
10153   create~extra~nodes,~
10154   create~medium~nodes,~
10155   create~large~nodes,~
10156   extra~left~margin,~
10157   extra~right~margin,~
10158   first~col,~
10159   first~row,~
10160   hlines,~
10161   hvlines,~

```

```

10162   hvlines-except-borders,~
10163   label,~
10164   last-col,~
10165   last-row,~
10166   left-margin,~
10167   light-syntax,~
10168   light-syntax-expanded,~
10169   name,~
10170   no-cell-nodes,~
10171   notes~(several~subkeys),~
10172   nullify-dots,~
10173   pgf-node-code,~
10174   renew-dots,~
10175   respect-arraystretch,~
10176   right-margin,~
10177   rounded-corners,~
10178   rules~(with~the~subkeys~'color'~and~'width'),~
10179   short-caption,~
10180   t,~
10181   tabularnote,~
10182   vlines,~
10183   xdots/color,~
10184   xdots/shorten-start,~
10185   xdots/shorten-end,~
10186   xdots/shorten-and~
10187   xdots/line-style.
10188 }

10189 \@@_msg_new:nnn { Duplicate-name }
10190 {
10191   Duplicate~name.\\
10192   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10193   the~same~environment~name~twice.~You~can~go~on,~but,~
10194   maybe,~you~will~have~incorrect~results~especially~
10195   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10196   message~again,~use~the~key~'allow-duplicate-names'~in~
10197   '\token_to_str:N \NiceMatrixOptions'.\\
10198   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10199     { For~a~list~of~the~names~already~used,~type-H~<return>. }
10200 }
10201 {
10202   The~names~already~defined~in~this~document~are:~
10203   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10204 }

10205 \@@_msg_new:nn { Option-auto-for-columns-width }
10206 {
10207   Erroneous~use.\\
10208   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10209   That~key~will~be~ignored.
10210 }

10211 \@@_msg_new:nn { NiceTabularX~without~X }
10212 {
10213   NiceTabularX~without~X.\\
10214   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10215   However,~you~can~go~on.
10216 }

10217 \@@_msg_new:nn { Preamble~forgotten }
10218 {
10219   Preamble~forgotten.\\
10220   You~have~probably~forgotten~the~preamble~of~your~
10221   \@@_full_name_env:. \\
10222   This~error~is~fatal.
10223 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	91
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
15	After the construction of the array	93
16	We draw the dotted lines	100
17	The actual instructions for drawing the dotted lines with Tikz	113
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	141
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	158
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	188
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

31	The keyword <code>\CodeAfter</code>	192
32	The delimiters in the preamble	192
33	The command <code>\SubMatrix</code>	194
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	202
35	The command <code>TikzEveryCell</code>	205
36	The command <code>\ShowCellNames</code>	207
37	We process the options at package loading	208
38	About the package underscore	210
39	Error messages of the package	210