

The code of the package `nicematrix`*

F. Pantigny
`fpantigny@wanadoo.fr`

January 6, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 7.0a of `nicematrix`, at the date of 2024/12/16.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

```

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

```

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

```

76 \NewDocumentCommand \@@_collect_options:nw { m r[] }
77 { \@@_collect_options:nn { #1 } { #2 } }
78
79 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
80 {
81   \peek_meaning:NTF [
82     { \@@_collect_options:nnw { #1 } { #2 } }
83     { #1 { #2 } }
84 }
85
86 \cs_new_protected:Npn \@@_collect_options:nnw #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_pgfulil@check@rerun \pgfutil@check@rerun

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

```

208     \skip_horizontal:N \c_zero_dim
209   }

```

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

214 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

278 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The following boolean will be raised 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 content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

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

```

697 \bool_new:N \l_@@_standard_cline_bool

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
717 \bool_new:N \l_@@_light_syntax_bool
718 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
730 \cs_new_protected:Npn \@@_reset_arraystretch:
731 { \cs_set_nopar:Npn \arraystretch { 1 } }
```

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

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

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

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

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

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

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

```
735 \bool_new:N \l_@@_medium_nodes_bool
736 \bool_new:N \l_@@_large_nodes_bool
```

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

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

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

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

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

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

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

```
742 \tl_new:N \l_@@_end_of_row_tl
743 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

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

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

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

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

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

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

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

```

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

```

The options down, up and middle are not documented for the final user because he should use the syntax with \wedge , $_$ and \therefore . 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 `\wedge{...}`.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

1044   renew-matrix .code:n = \@@_renew_matrix: ,
1045   renew-matrix .value_forbidden:n = true ,

```

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

```

1046   exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

If the option `columns-width` is used, all the columns will have the same width.

In `\NiceMatrixOptions`, the special value `auto` is not available.

```

1047   columns-width .code:n =

```


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

```

1048 \str_if_eq:eeTF { #1 } { auto }
1049 { \@@_error:n { Option~auto~for~columns~width } }
1050 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

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

```

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

```

1064 \NewDocumentCommand \NiceMatrixOptions { m }
1065 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

8 Important code used by {NiceArrayWithDelims}

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

```

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

```

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

```
1152   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1154   \int_gincr:N \c@jCol
```

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

```

1155   \int_compare:nNnT \c@jCol = \c_one_int
1156     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }

```

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

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

The following command is nullified in the tabulars.

```
1158   \@@_tuning_not_tabular_begin:
```

```
1159   \@@_tuning_first_row:
```

```
1160   \@@_tuning_last_row:
```

```
1161   \g_@@_row_style_tl
```

```
1162 }

```

The following command will be nullified unless there is a first row.

Here is a version with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_tuning_first_row:
{
  \int_if_zero:nT \c@iRow
  {
    \int_compare:nNnT \c@jCol > 0
    {
      \l_@@_code_for_first_row_tl
    }
  }
}

```

```

        \xglobal \colorlet { nicematrix-first-row } { . }
    }
}

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

The following commands are nullified in the tabulars.

```

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

```

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

```

1184     \@@_tuning_key_small:
1185 }
1186 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token

```

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1286 \@@_update_max_cell_width:

```

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

```

1287 \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

1365 #1
1366 }
1367 \box_use:N \l_@@_cell_box
1368 \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1369 \hbox_overlap_left:n
1370 {
1371 \pgfsys@markposition
1372 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1373 #1
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     \pgfrememberpicturerepositiononpagetrue
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 theses 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:NnT
1463       \l_@@_hlines_clist
1464       { \int_eval:n { \c@iRow + 1 } }
1465     }
1466   }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁵ and `\extrarowheight`

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

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

(of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that's why we do it in the `\ialign`.

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

```

1595 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1596 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1597 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1598 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1599 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1600 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1601 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1602 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1603 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1604 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1605 \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:`.

```

1606 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1607 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1608 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1609 \@@_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!).

```

1610 \tl_if_exist:NT \l_@@_note_in_caption_tl
1611 {
1612   \tl_if_empty:NF \l_@@_note_in_caption_tl
1613   {
1614     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1615     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1616   }
1617 }

```

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).

```

1618 \seq_gclear:N \g_@@_multicolumn_cells_seq
1619 \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).

```

1620 \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`).

```

1621 \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.

```

1622 \int_gzero_new:N \g_@@_col_total_int
1623 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1624 \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.

```

1625 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1626 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1627 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1628 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1629 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1630 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

```

```

1631 \tl_gclear:N \g_nicematrix_code_before_tl
1632 \tl_gclear:N \g_@@_pre_code_before_tl
1633 }

```

This is the end of \@@_pre_array_ii:.

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

```

1634 \cs_new_protected:Npn \@@_pre_array:
1635 {
1636   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1637   \int_gzero_new:N \c@iRow
1638   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1639   \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).

```

1640   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1641   {
1642     \bool_set_true:N \l_@@_last_row_without_value_bool
1643     \bool_if:NT \g_@@_aux_found_bool
1644       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1645   }
1646   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1647   {
1648     \bool_if:NT \g_@@_aux_found_bool
1649       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1650   }

```

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”.

```

1651   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1652   {
1653     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1654     {
1655       \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1656       { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1657       \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1658       { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1659     }
1660   }

1661   \seq_gclear:N \g_@@_cols_vlism_seq
1662   \seq_gclear:N \g_@@_submatrix_seq

```

Now the \CodeBefore.

```

1663   \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.

```

1664   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1665   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1666   \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.

```
1667 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1668 \@@_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.

```
1669 \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]`).

```
1670 \dim_zero_new:N \l_@@_left_delim_dim
1671 \dim_zero_new:N \l_@@_right_delim_dim
1672 \bool_if:NTF \g_@@_delims_bool
1673 {
```

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

```
1674 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1675 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1676 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1677 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1678 }
1679 {
1680 \dim_gset:Nn \l_@@_left_delim_dim
1681 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1682 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1683 }
```

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).

```
1684 \hbox_set:Nw \l_@@_the_array_box
1685 \skip_horizontal:N \l_@@_left_margin_dim
1686 \skip_horizontal:N \l_@@_extra_left_margin_dim
1687 \bool_if:NT \c_@@_recent_array_bool
1688 { \UseTaggingSocket { tbl / hmode / begin } }
```

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

```
1689 \m@th
1690 \c_math_toggle_token
1691 \bool_if:NTF \l_@@_light_syntax_bool
1692 { \use:c { @@-light-syntax } }
1693 { \use:c { @@-normal-syntax } }
1694 }
```

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

```
1695 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1696 {
1697 \tl_set:Nn \l_tmpa_tl { #1 }
1698 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1699 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1700 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1701 \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...

```
1702   \@@_pre_array:
1703 }
```

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).

```
1704 \cs_new_protected:Npn \@@_pre_code_before:
1705 {
```

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).

```
1706   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1707   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1708   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1709   \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

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

```
1710   \pgfsys@markposition { \@@_env: - position }
1711   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1712   \pgfpicture
1713   \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1714   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1715   {
1716     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1717     \pgfcoordinate { \@@_env: - row - ##1 }
1718     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1719   }
```

Now, the recreation of the `col` nodes.

```
1720   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1721   {
1722     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1723     \pgfcoordinate { \@@_env: - col - ##1 }
1724     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1725   }
```

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

```
1726   \@@_create_diag_nodes:
```

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

```
1727   \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1728   \endpgfpicture
```

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

```
1729   \@@_create_blocks_nodes:
```

```

1730 \IfPackageLoadedT { tikz }
1731 {
1732   \tikzset
1733   {
1734     every-picture / .style =
1735     { overlay , name-prefix = \@@_env: - }
1736   }
1737 }
1738 \cs_set_eq:NN \cellcolor \@@_cellcolor
1739 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1740 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1741 \cs_set_eq:NN \rowcolor \@@_rowcolor
1742 \cs_set_eq:NN \rowcolors \@@_rowcolors
1743 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1744 \cs_set_eq:NN \arraycolor \@@_arraycolor
1745 \cs_set_eq:NN \columncolor \@@_columncolor
1746 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1747 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1748 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1749 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1750 }

```

```

1751 \cs_new_protected:Npn \@@_exec_code_before:
1752 {

```

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...

```

1753 \clist_map_inline:Nn \l_@@_corners_cells_clist
1754 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1755 \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`.

```

1756 \@@_add_to_colors_seq:nn { { nocolor } } { }
1757 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1758 \group_begin:

```

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

```

1759 \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`).

```

1760 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1761 { \@@_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`.

```

1762 \exp_last_unbraced:No \@@_CodeBefore_keys:
1763 \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.

```

1764 \@@_actually_color:
1765 \l_@@_code_before_tl
1766 \q_stop

```

```

1767 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1768 \group_end:
1769 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1770 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1771 }

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

1782 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1783 {
1784   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1785   \@@_CodeBefore:w
1786 }

```

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.

```

1787 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1788 {
1789   \bool_if:NT \g_@@_aux_found_bool
1790   {
1791     \@@_pre_code_before:
1792     #1
1793   }
1794 }

```

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.

```

1795 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1796 {
1797   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1798   {
1799     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1800     \pgfcoordinate { \@@_env: - row - ##1 - base }
1801     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1802     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1803     {
1804       \cs_if_exist:cT
1805       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1806       {
1807         \pgfsys@getposition
1808         { \@@_env: - ##1 - #####1 - NW }
1809         \@@_node_position:
1810         \pgfsys@getposition
1811         { \@@_env: - ##1 - #####1 - SE }
1812         \@@_node_position_i:
1813         \@@_pgf_rect_node:nnn
1814         { \@@_env: - ##1 - #####1 }
1815         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1816         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1817       }
1818     }
1819 }

```

```

1819     }
1820     \int_step_inline:nn \c@iRow
1821     {
1822         \pgfnodealias
1823         { \@@_env: - ##1 - last }
1824         { \@@_env: - ##1 - \int_use:N \c@jCol }
1825     }
1826     \int_step_inline:nn \c@jCol
1827     {
1828         \pgfnodealias
1829         { \@@_env: - last - ##1 }
1830         { \@@_env: - \int_use:N \c@iRow - ##1 }
1831     }
1832     \@@_create_extra_nodes:
1833 }

```

```

1834 \cs_new_protected:Npn \@@_create_blocks_nodes:
1835 {
1836     \pgfpicture
1837     \pgf@relevantforpicturesizefalse
1838     \pgfrememberpicturepositiononpagetrue
1839     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1840     { \@@_create_one_block_node:nnnnn ##1 }
1841     \endpgfpicture
1842 }

```

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

```

1843 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1844 {
1845     \tl_if_empty:nF { #5 }
1846     {
1847         \@@_qpoint:n { col - #2 }
1848         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1849         \@@_qpoint:n { #1 }
1850         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1851         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1852         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1853         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1854         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1855         \@@_pgf_rect_node:nnnnn
1856         { \@@_env: - #5 }
1857         { \dim_use:N \l_tmpa_dim }
1858         { \dim_use:N \l_tmpb_dim }
1859         { \dim_use:N \l_@@_tmpc_dim }
1860         { \dim_use:N \l_@@_tmpd_dim }
1861     }
1862 }

```

```

1863 \cs_new_protected:Npn \@@_patch_for_revtext:
1864 {
1865     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1866     \cs_set_eq:NN \@array \@array@array
1867     \cs_set_eq:NN \@tabular \@tabular@array
1868     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1869     \cs_set_eq:NN \array \array@array
1870     \cs_set_eq:NN \endarray \endarray@array
1871     \cs_set:Npn \endtabular { \endarray $\egroup} % $

```

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

```

1872 \cs_set_eq:NN \@mkpream \@mkpream@array
1873 \cs_set_eq:NN \@classx \@classx@array
1874 \cs_set_eq:NN \@insert@column \@insert@column@array
1875 \cs_set_eq:NN \@arraycr \@arraycr@array
1876 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1877 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1878 }

```

10 The environment `{NiceArrayWithDelims}`

```

1879 \NewDocumentEnvironment { NiceArrayWithDelims }
1880 { m m O { } m ! O { } t \CodeBefore }
1881 {
1882   \bool_if:NT \c_@@_revtex_bool \@_patch_for_revtex:
1883   \@_provide_pgfsyspdfmark:
1884   \bool_if:NT \g_@@_footnote_bool \savenotes

```

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

```

1885   \bgroup

1886   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1887   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1888   \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1889   \tl_if_empty:NT \g_@@_user_preamble_tl { \@_fatal:n { empty~preamble } }

1890   \int_gzero:N \g_@@_block_box_int
1891   \dim_zero:N \g_@@_width_last_col_dim
1892   \dim_zero:N \g_@@_width_first_col_dim
1893   \bool_gset_false:N \g_@@_row_of_col_done_bool
1894   \str_if_empty:NT \g_@@_name_env_str
1895   { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1896   \bool_if:NTF \l_@@_tabular_bool
1897     \mode_leave_vertical:
1898     \@_test_if_math_mode:
1899   \bool_if:NT \l_@@_in_env_bool { \@_fatal:n { Yet~in~env } }
1900   \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.

```

1901   \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.

```

1902   \cs_if_exist:NT \tikz@library@external@loaded
1903   {
1904     \tikzexternaldisable
1905     \cs_if_exist:NT \ifstandalone
1906     { \tikzset { external / optimize = false } }
1907   }

```

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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1908 \int_gincr:N \g_@@_env_int
1909 \bool_if:NF \l_@@_block_auto_columns_width_bool
1910 { \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`).

```

1911 \seq_gclear:N \g_@@_blocks_seq
1912 \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`.

```

1913 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1914 \seq_gclear:N \g_@@_pos_of_xdots_seq
1915 \tl_gclear_new:N \g_@@_code_before_tl
1916 \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.

```

1917 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1918 {
1919   \bool_gset_true:N \g_@@_aux_found_bool
1920   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1921 }
1922 { \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.

```

1923 \tl_gclear:N \g_@@_aux_tl
1924 \tl_if_empty:NF \g_@@_code_before_tl
1925 {
1926   \bool_set_true:N \l_@@_code_before_bool
1927   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1928 }
1929 \tl_if_empty:NF \g_@@_pre_code_before_tl
1930 { \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`.

```

1931 \bool_if:NTF \g_@@_delims_bool
1932 { \keys_set:nn { nicematrix / pNiceArray } }
1933 { \keys_set:nn { nicematrix / NiceArray } }
1934 { #3 , #5 }

```

```

1935 \@@_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:.`

```

1936 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1937 }

```

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

```

1938 {
1939   \bool_if:NTF \l_@@_light_syntax_bool
1940   { \use:c { end @@-light-syntax } }
1941   { \use:c { end @@-normal-syntax } }
1942   \c_math_toggle_token
1943   \skip_horizontal:N \l_@@_right_margin_dim
1944   \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1945
1946 % awful workaround
1947 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1948 {
1949     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1950     {
1951         \skip_horizontal:N - \l_@@_columns_width_dim
1952         \bool_if:NTF \l_@@_tabular_bool
1953             { \skip_horizontal:n { - 2 \tabcolsep } }
1954             { \skip_horizontal:n { - 2 \arraycolsep } }
1955     }
1956 }
1957 \hbox_set_end:
1958 \bool_if:NT \c_@@_recent_array_bool
1959 { \UseTaggingSocket { tbl / hmode / end } }

```

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

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

```

1960 \bool_if:NT \l_@@_width_used_bool
1961 {
1962     \int_if_zero:nT \g_@@_total_X_weight_int
1963     { \@@_error_or_warning:n { width~without~X~columns } }
1964 }

```

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 .

```

1965 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1966 { \@@_compute_width_X: }

```

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

```

1967 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1968 {
1969     \bool_if:NF \l_@@_last_row_without_value_bool
1970     {
1971         \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1972         {
1973             \@@_error:n { Wrong~last~row }
1974             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1975         }
1976     }
1977 }

```

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

```

1978 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1979 \bool_if:NTF \g_@@_last_col_found_bool
1980 { \int_gdecr:N \c@jCol }
1981 {
1982     \int_compare:nNnT \l_@@_last_col_int > { -1 }
1983     { \@@_error:n { last~col~not~used } }
1984 }

```

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

```

1985 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1986 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

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

```

1987 \int_if_zero:nTF \l_@@_first_col_int
1988 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

1989 \bool_if:nTF { ! \g_@@_delims_bool }
1990 {
1991   \str_if_eq:eeTF \l_@@_baseline_tl { c }
1992   \@@_use_arraybox_with_notes_c:
1993   {
1994     \str_if_eq:eeTF \l_@@_baseline_tl { b }
1995     \@@_use_arraybox_with_notes_b:
1996     \@@_use_arraybox_with_notes:
1997   }
1998 }

```

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).

```

1999 {
2000   \int_if_zero:nTF \l_@@_first_row_int
2001   {
2002     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2003     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2004   }
2005   { \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

```

2006   \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2007   {
2008     \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2009     \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2010   }
2011   { \dim_zero:N \l_tmpb_dim }
2012   \hbox_set:Nn \l_tmpa_box
2013   {
2014     \m@th % added 2024/11/21
2015     \c_math_toggle_token
2016     \@@_color:o \l_@@_delimiters_color_tl
2017     \exp_after:wN \left \g_@@_left_delim_tl
2018     \vcenter
2019     {

```

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.

```

2020       \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2021       \hbox
2022       {
2023         \bool_if:NTF \l_@@_tabular_bool
2024         { \skip_horizontal:N -\tabcolsep }
2025         { \skip_horizontal:N -\arraycolsep }
2026         \@@_use_arraybox_with_notes_c:
2027         \bool_if:NTF \l_@@_tabular_bool
2028         { \skip_horizontal:N -\tabcolsep }
2029         { \skip_horizontal:N -\arraycolsep }
2030       }

```

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

```

2031       \skip_vertical:N -\l_tmpb_dim
2032       \skip_vertical:N \arrayrulewidth

```

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


```

2033     }
2034     \exp_after:wN \right \g_@@_right_delim_tl
2035     \c_math_toggle_token
2036 }

```

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.

```

2037     \bool_if:NTF \l_@@_delimiters_max_width_bool
2038     {
2039         \@@_put_box_in_flow_bis:nn
2040         \g_@@_left_delim_tl
2041         \g_@@_right_delim_tl
2042     }
2043     \@@_put_box_in_flow:
2044 }

```

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).

```

2045     \bool_if:NT \g_@@_last_col_found_bool
2046     { \skip_horizontal:N \g_@@_width_last_col_dim }
2047     \bool_if:NT \l_@@_preamble_bool
2048     {
2049         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2050         { \@_warning_gredirect_none:n { columns-not-used } }
2051     }
2052     \@@_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.

```

2053     \egroup

```

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

```

2054     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2055     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2056     \iow_now:Ne \@mainaux
2057     {
2058         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2059         { \exp_not:o \g_@@_aux_tl }
2060     }
2061     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2062     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2063 }

```

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

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

```

2064     \cs_new_protected:Npn \@@_compute_width_X:
2065     {
2066         \tl_gput_right:Ne \g_@@_aux_tl
2067         {
2068             \bool_set_true:N \l_@@_X_columns_aux_bool
2069             \dim_set:Nn \l_@@_X_columns_dim
2070             {
2071                 \dim_compare:nNnTF
2072                 {
2073                     \dim_abs:n
2074                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2075                 }
2076                 <

```

```

2077         { 0.001 pt }
2078         { \dim_use:N \l_@@_X_columns_dim }
2079         {
2080             \dim_eval:n
2081             {
2082                 ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2083                 / \int_use:N \g_@@_total_X_weight_int
2084                 + \l_@@_X_columns_dim
2085             }
2086         }
2087     }
2088 }
2089 }

```

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.

```

2090 \cs_new_protected:Npn \@@_transform_preamble:
2091 {
2092     \@@_transform_preamble_i:
2093     \@@_transform_preamble_ii:
2094 }
2095 \cs_new_protected:Npn \@@_transform_preamble_i:
2096 {
2097     \int_gzero:N \c@jCol

```

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

```

2098     \seq_gclear:N \g_@@_cols_vlsim_seq

```

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble provided by the final user.

```

2099     \bool_gset_false:N \g_tmpb_bool

```

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

```

2100     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2101     \int_zero:N \l_tmpa_int
2102     \tl_gclear:N \g_@@_array_preamble_tl
2103     \str_if_eq:eeTF \l_@@_vlines_clist { all }
2104     {
2105         \tl_gset:Nn \g_@@_array_preamble_tl
2106         { ! { \skip_horizontal:N \arrayrulewidth } }
2107     }
2108     {
2109         \clist_if_in:NnT \l_@@_vlines_clist 1
2110         {
2111             \tl_gset:Nn \g_@@_array_preamble_tl
2112             { ! { \skip_horizontal:N \arrayrulewidth } }
2113         }
2114     }

```

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

```

2115     \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2116     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

```

2117 \@@_replace_columncolor:
2118 }

```

```

2119 \hook_gput_code:nnn { begindocument } { . }
2120 {
2121   \IfPackageLoadedTF { colortbl }
2122   {

```

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).

```

2123     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2124     \cs_new_protected:Npn \@@_replace_columncolor:
2125     {
2126       \regex_replace_all:NnN
2127       \c_@@_columncolor_regex
2128       { \c { @@_columncolor_preamble } }
2129       \g_@@_array_preamble_tl
2130     }
2131   }
2132   {
2133     \cs_new_protected:Npn \@@_replace_columncolor:
2134     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2135   }
2136 }

```

```

2137 \cs_new_protected:Npn \@@_transform_preamble_ii:
2138 {

```

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

```

2139   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2140   {
2141     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2142     { \bool_gset_true:N \g_@@_delims_bool }
2143   }
2144   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2145   \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).

```

2146   \int_if_zero:nTF \l_@@_first_col_int
2147   { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2148   {
2149     \bool_if:NF \g_@@_delims_bool
2150     {
2151       \bool_if:NF \l_@@_tabular_bool
2152       {
2153         \clist_if_empty:NT \l_@@_vlines_clist
2154         {
2155           \bool_if:NF \l_@@_exterior_arraycolsep_bool
2156           { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2157         }
2158       }
2159     }
2160   }
2161   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2162   { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2163   {
2164     \bool_if:NF \g_@@_delims_bool

```

```

2165     {
2166       \bool_if:NF \l_@@_tabular_bool
2167       {
2168         \clist_if_empty:NT \l_@@_vlines_clist
2169         {
2170           \bool_if:NF \l_@@_exterior_arraycolsep_bool
2171           { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2172         }
2173       }
2174     }
2175   }

```

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`).

```

2176   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2177   {

```

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.

```

2178     \bool_if:NF \c_@@_testphase_table_bool
2179     {
2180       \tl_gput_right:Nn \g_@@_array_preamble_tl
2181       { > { \@@_error_too_much_cols: } l }
2182     }
2183   }
2184 }

```

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.

```

2185 \cs_new_protected:Npn \@@_rec_preamble:n #1
2186 {

```

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

```

2187   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2188   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2189   {

```

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

```

2190     \cs_if_exist:cTF { NC @ find @ #1 }
2191     {
2192       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2193       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2194     }
2195     {
2196       \str_if_eq:nnTF { #1 } { S }
2197       { \@@_fatal:n { unknown~column~type~S } }
2198       { \@@_fatal:nn { unknown~column~type } { #1 } }
2199     }
2200   }
2201 }

```

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

```

2202 \cs_new_protected:Npn \@@_c #1
2203 {
2204   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl

```

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

```

2205 \tl_gclear:N \g_@@_pre_cell_tl
2206 \tl_gput_right:Nn \g_@@_array_preamble_tl
2207 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2208 \int_gincr:N \c@jCol
2209 \@@_rec_preamble_after_col:n
2210 }
2211 \cs_new_protected:Npn \@@_l #1
2212 {
2213 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2214 \tl_gclear:N \g_@@_pre_cell_tl
2215 \tl_gput_right:Nn \g_@@_array_preamble_tl
2216 {
2217 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2218 l
2219 < \@@_cell_end:
2220 }
2221 \int_gincr:N \c@jCol
2222 \@@_rec_preamble_after_col:n
2223 }
2224 \cs_new_protected:Npn \@@_r #1
2225 {
2226 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2227 \tl_gclear:N \g_@@_pre_cell_tl
2228 \tl_gput_right:Nn \g_@@_array_preamble_tl
2229 {
2230 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2231 r
2232 < \@@_cell_end:
2233 }
2234 \int_gincr:N \c@jCol
2235 \@@_rec_preamble_after_col:n
2236 }

```

For ! and @

```

2237 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2238 {
2239 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2240 \@@_rec_preamble:n
2241 }
2242 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2243 \cs_new_protected:cpn { @@ _ | } #1
2244 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2245 \int_incr:N \l_tmpa_int
2246 \@@_make_preamble_i_i:n
2247 }
2248 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2249 {
2250 \str_if_eq:nnTF { #1 } { | }
2251 { \use:c { @@ _ | } | }
2252 { \@@_make_preamble_i_ii:nn { } #1 }
2253 }
2254 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2255 {
2256 \str_if_eq:nnTF { #2 } { [ ] }
2257 { \@@_make_preamble_i_iii:nn { #1 } [ ] }
2258 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }

```

```

2259 }
2260 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2261 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2262 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2263 {
2264   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2265   \tl_gput_right:Ne \g_@@_array_preamble_tl
2266   {

```

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

```

2267     \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2268   }
2269   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2270   {
2271     \@@_vline:n
2272     {
2273       position = \int_eval:n { \c@jCol + 1 } ,
2274       multiplicity = \int_use:N \l_tmpa_int ,
2275       total-width = \dim_use:N \l_@@_rule_width_dim ,
2276       #2
2277     }

```

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.

```

2278   }
2279   \int_zero:N \l_tmpa_int
2280   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2281   \@@_rec_preamble:n #1
2282 }

2283 \cs_new_protected:cpn { @@ _ > } #1 #2
2284 {
2285   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2286   \@@_rec_preamble:n
2287 }

2288 \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.

```

2289 \keys_define:nn { nicematrix / p-column }
2290 {
2291   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2292   r .value_forbidden:n = true ,
2293   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2294   c .value_forbidden:n = true ,
2295   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2296   l .value_forbidden:n = true ,
2297   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2298   S .value_forbidden:n = true ,
2299   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2300   p .value_forbidden:n = true ,
2301   t .meta:n = p ,
2302   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2303   m .value_forbidden:n = true ,
2304   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2305   b .value_forbidden:n = true
2306 }

```

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

```

2307 \cs_new_protected:Npn \@@_p #1
2308 {
2309   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2310 \@@_make_preamble_ii_i:n
2311 }
2312 \cs_set_eq:NN \@@_b \@@_p
2313 \cs_set_eq:NN \@@_m \@@_p
2314 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2315 {
2316   \str_if_eq:nnTF { #1 } { [ ] }
2317   { \@@_make_preamble_ii_ii:w [ ] }
2318   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2319 }
2320 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2321 { \@@_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.

```

2322 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2323 {

```

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).

```

2324   \str_set:Nn \l_@@_hpos_col_str { j }
2325   \@@_keys_p_column:n { #1 }
2326   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2327 }
2328 \cs_new_protected:Npn \@@_keys_p_column:n #1
2329 { \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.

```

2330 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2331 {
2332   \use:e
2333   {
2334     \@@_make_preamble_ii_v:nnnnnnnn
2335     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2336     { \dim_eval:n { #1 } }
2337   }

```

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.

```

2338   \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2339   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2340   {

```

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

```

2341     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2342     { \str_lowercase:o \l_@@_hpos_col_str }
2343   }
2344   \IfPackageLoadedTF { ragged2e }
2345   {
2346     \str_case:on \l_@@_hpos_col_str
2347     {
2348       c { \exp_not:N \Centering }
2349       l { \exp_not:N \RaggedRight }
2350       r { \exp_not:N \RaggedLeft }
2351     }
2352   }
2353   {
2354     \str_case:on \l_@@_hpos_col_str

```

```

2355         {
2356             c { \exp_not:N \centering }
2357             l { \exp_not:N \raggedright }
2358             r { \exp_not:N \raggedleft }
2359         }
2360     }
2361     #3
2362 }
2363 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2364 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2365 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2366 { #2 }
2367 {
2368     \str_case:onF \l_@@_hpos_col_str
2369     {
2370         { j } { c }
2371         { si } { c }
2372     }

```

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

```

2373     { \str_lowercase:o \l_@@_hpos_col_str }
2374 }
2375 }

```

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

```

2376     \int_gincr:N \c@jCol
2377     \@@_rec_preamble_after_col:n
2378 }

```

#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*.

```

2379 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2380 {
2381     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2382     {
2383         \tl_gput_right:Nn \g_@@_array_preamble_tl
2384         { > \@@_test_if_empty_for_S: }
2385     }
2386     {
2387         \tl_gput_right:Nn \g_@@_array_preamble_tl
2388         { > \@@_test_if_empty: }
2389     }
2390     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2391     \tl_gclear:N \g_@@_pre_cell_tl
2392     \tl_gput_right:Nn \g_@@_array_preamble_tl
2393     {
2394         > {

```

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.

```

2395         \dim_set:Nn \l_@@_col_width_dim { #2 }
2396         \bool_if:NT \c_@@_testphase_table_bool

```



```

2397         { \tag_struct_begin:n { tag = Div } }
2398     \@@_cell_begin:

```

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

```

2399         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2400     \everypar
2401     {
2402         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2403     \everypar { }
2404     }
2405     \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.).

```

2406         #3

```

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

```

2407     \g_@@_row_style_tl
2408     \arraybackslash
2409     #5
2410 }
2411 #8
2412 < {
2413     #6

```

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

```

2414     \@finalstrut \@arstrutbox
2415     \use:c { end #7 }

```

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

```

2416         #4
2417     \@@_cell_end:
2418     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2419 }
2420 }
2421 }

```

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

```

2422 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2423 {

```

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 `&`).

```

2424 \group_align_safe_begin:
2425 \peek_meaning:NTF &
2426 {
2427     \group_align_safe_end:
2428     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2429     {

```

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

```

2430         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2431         \skip_horizontal:N \l_@@_col_width_dim
2432     }
2433 }
2434 { \group_align_safe_end: }
2435 }

```

```

2436 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2437 {
2438   \peek_meaning:NT \__siunitx_table_skip:n
2439   { \bool_gset_true:N \g_@@_empty_cell_bool }
2440 }

```

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 than the height of `\strutbox`, there is only one row.

```

2441 \cs_new_protected:Npn \@@_center_cell_box:
2442 {

```

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

```

2443   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2444   {
2445     \int_compare:nNnT
2446     { \box_ht:N \l_@@_cell_box }
2447     >

```

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).

```

2448     { \box_ht:N \strutbox }
2449     {
2450       \hbox_set:Nn \l_@@_cell_box
2451       {
2452         \box_move_down:nn
2453         {
2454           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2455             + \baselineskip ) / 2
2456         }
2457         { \box_use:N \l_@@_cell_box }
2458       }
2459     }
2460   }
2461 }

```

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

```

2462 \cs_new_protected:Npn \@@_V #1 #2
2463 {
2464   \str_if_eq:nnTF { #1 } { [ ] }
2465   { \@@_make_preamble_V_i:w [ ] }
2466   { \@@_make_preamble_V_i:w [ ] { #2 } }
2467 }
2468 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2469 { \@@_make_preamble_V_ii:nn { #1 } }
2470 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2471 {
2472   \str_set:Nn \l_@@_vpos_col_str { p }
2473   \str_set:Nn \l_@@_hpos_col_str { j }
2474   \@@_keys_p_column:n { #1 }
2475   \IfPackageLoadedTF { varwidth }
2476   { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2477   {
2478     \@@_error_or_warning:n { varwidth-not-loaded }
2479     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2480   }
2481 }

```

For w and W

```
2482 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2483 \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.

```
2484 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2485 {
2486   \str_if_eq:nnTF { #3 } { s }
2487   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2488   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2489 }
```

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.

```
2490 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2491 {
2492   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2493   \tl_gclear:N \g_@@_pre_cell_tl
2494   \tl_gput_right:Nn \g_@@_array_preamble_tl
2495   {
2496     > {
2497       \dim_set:Nn \l_@@_col_width_dim { #2 }
2498       \@@_cell_begin:
2499       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2500     }
2501     c
2502     < {
2503       \@@_cell_end_for_w_s:
2504       #1
2505       \@@_adjust_size_box:
2506       \box_use_drop:N \l_@@_cell_box
2507     }
2508   }
2509   \int_gincr:N \c_jCol
2510   \@@_rec_preamble_after_col:n
2511 }
```

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

```
2512 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2513 {
2514   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2515   \tl_gclear:N \g_@@_pre_cell_tl
2516   \tl_gput_right:Nn \g_@@_array_preamble_tl
2517   {
2518     > {
```

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.

```
2519       \dim_set:Nn \l_@@_col_width_dim { #4 }
2520       \hbox_set:Nw \l_@@_cell_box
2521       \@@_cell_begin:
2522       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2523     }
2524     c
2525     < {
2526       \@@_cell_end:
2527       \hbox_set_end:
2528       #1
```

```

2529         \@@_adjust_size_box:
2530         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2531     }
2532 }

```

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

```

2533     \int_gincr:N \c@jCol
2534     \@@_rec_preamble_after_col:n
2535 }

```

```

2536 \cs_new_protected:Npn \@@_special_W:
2537 {
2538     \dim_compare:nNtT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2539     { \@@_warning:n { W~warning } }
2540 }

```

For S (of siunitx).

```

2541 \cs_new_protected:Npn \@@_S #1 #2
2542 {
2543     \str_if_eq:nnTF { #2 } { [ ] }
2544     { \@@_make_preamble_S:w [ ] }
2545     { \@@_make_preamble_S:w [ ] { #2 } }
2546 }
2547 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2548 { \@@_make_preamble_S_i:n { #1 } }
2549 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2550 {
2551     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2552     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2553     \tl_gclear:N \g_@@_pre_cell_tl
2554     \tl_gput_right:Nn \g_@@_array_preamble_tl
2555     {
2556         > {
2557             \@@_cell_begin:
2558             \keys_set:nn { siunitx } { #1 }
2559             \siunitx_cell_begin:w
2560         }
2561         c
2562         < { \siunitx_cell_end: \@@_cell_end: }
2563     }

```

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

```

2564     \int_gincr:N \c@jCol
2565     \@@_rec_preamble_after_col:n
2566 }

```

For (, [and \{.

```

2567 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2568 {
2569     \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.

```

2570     \int_if_zero:nTF \c@jCol
2571     {
2572         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2573         {

```

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

```

2574         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2575         \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2576         \@@_rec_preamble:n #2
2577     }

```

```

2578     {
2579         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2580         \@@_make_preamble_iv:nn { #1 } { #2 }
2581     }
2582 }
2583 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2584 }
2585 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2586 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2587 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2588 {
2589     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2590     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2591     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2592     {
2593         \@@_error:nn { delimiter~after~opening } { #2 }
2594         \@@_rec_preamble:n
2595     }
2596     { \@@_rec_preamble:n #2 }
2597 }

```

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

```

2598 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2599 { \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}`).

```

2600 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2601 {
2602     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2603     \tl_if_in:nnTF { ) ] \} } { #2 }
2604     { \@@_make_preamble_v:nnn #1 #2 }
2605     {
2606         \str_if_eq:nnTF { \@@_stop: } { #2 }
2607         {
2608             \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2609             { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2610             {
2611                 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2612                 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2613                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2614                 \@@_rec_preamble:n #2
2615             }
2616         }
2617         {
2618             \tl_if_in:nnT { ( [ \{ \left } { #2 }
2619             { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2620             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2621             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2622             \@@_rec_preamble:n #2
2623         }
2624     }
2625 }
2626 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2627 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2628 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2629 {
2630     \str_if_eq:nnTF { \@@_stop: } { #3 }
2631     {
2632         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl

```

```

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       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2638     }
2639     {
2640       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2641       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2642       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2643       \@@_error:nn { double~closing~delimiter } { #2 }
2644     }
2645   }
2646   {
2647     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2648     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2649     \@@_error:nn { double~closing~delimiter } { #2 }
2650     \@@_rec_preamble:n #3
2651   }
2652 }

2653 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2654 { \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 @{...}.

```

2655 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2656 {
2657   \str_if_eq:nnTF { #1 } { < }
2658   \@@_rec_preamble_after_col_i:n
2659   {
2660     \str_if_eq:nnTF { #1 } { @ }
2661     \@@_rec_preamble_after_col_ii:n
2662     {
2663       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2664       {
2665         \tl_gput_right:Nn \g_@@_array_preamble_tl
2666         { ! { \skip_horizontal:N \arrayrulewidth } }
2667       }
2668       {
2669         \clist_if_in:NeT \l_@@_vlines_clist
2670         { \int_eval:n { \c@jCol + 1 } }
2671         {
2672           \tl_gput_right:Nn \g_@@_array_preamble_tl
2673           { ! { \skip_horizontal:N \arrayrulewidth } }
2674         }
2675       }
2676       \@@_rec_preamble:n { #1 }
2677     }
2678   }
2679 }

2680 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2681 {
2682   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2683   \@@_rec_preamble_after_col:n
2684 }

```

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.

```

2685 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2686 {

```

```

2687 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2688 {
2689   \tl_gput_right:Nn \g_@@_array_preamble_tl
2690   { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2691 }
2692 {
2693   \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2694   {
2695     \tl_gput_right:Nn \g_@@_array_preamble_tl
2696     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2697   }
2698   { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2699 }
2700 \@@_rec_preamble:n
2701 }

2702 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2703 {
2704   \tl_clear:N \l_tmpa_tl
2705   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2706   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2707 }

```

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

```

2708 \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.

```

2709 \cs_new_protected:Npn \@@_X #1 #2
2710 {
2711   \str_if_eq:nnTF { #2 } { [ ]
2712     { \@@_make_preamble_X:w [ ]
2713       { \@@_make_preamble_X:w [ ] #2 }
2714     }
2715   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2716   { \@@_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`).

```

2717 \keys_define:nn { nicematrix / X-column }
2718 { 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.

```

2719 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2720 {

```

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).

```

2721   \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).

```

2722   \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`.

```
2723 \int_zero_new:N \l_@@_weight_int
2724 \int_set_eq:NN \l_@@_weight_int \c_one_int
2725 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2726 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2727 \int_compare:nNtT \l_@@_weight_int < \c_zero_int
2728 {
2729   \@@_error_or_warning:n { negative-weight }
2730   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2731 }
2732 \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).

```
2733 \bool_if:NTF \l_@@_X_columns_aux_bool
2734 {
2735   \@@_make_preamble_ii_iv:nnn
2736   { \l_@@_weight_int \l_@@_X_columns_dim }
2737   { minipage }
2738   { \@@_no_update_width: }
2739 }
2740 {
2741   \tl_gput_right:Nn \g_@@_array_preamble_tl
2742   {
2743     > {
2744       \@@_cell_begin:
2745       \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`.

```
2746 \NotEmpty
```

The following code will nullify the box of the cell.

```
2747 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2748 { \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`.

```
2749 \begin { minipage } { 5 cm } \arraybackslash
2750 }
2751 c
2752 < {
2753   \end { minipage }
2754   \@@_cell_end:
2755 }
2756 }
2757 \int_gincr:N \c@jCol
2758 \@@_rec_preamble_after_col:n
2759 }
2760 }
```

```
2761 \cs_new_protected:Npn \@@_no_update_width:
2762 {
2763   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2764   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2765 }
```


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

```

2766 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2767 {
2768   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2769   { \int_eval:n { \c@jCol + 1 } }
2770   \tl_gput_right:Ne \g_@@_array_preamble_tl
2771   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2772   \@@_rec_preamble:n
2773 }

```

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.

```

2774 \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).

```

2775 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2776 { \@@_fatal:n { Preamble-forgotten } }
2777 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2778 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2779 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2780 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2781 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2782 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```

2783 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2784 {

```

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`.

```

2785   \multispan { #1 }
2786   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2787   \begingroup
2788   \bool_if:NT \c_@@_testphase_table_bool
2789   { \tbl_update_multicolumn_cell_data:n { #1 } }
2790   \cs_set_nopar:Npn \@addamp
2791   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2792   \tl_gclear:N \g_@@_preamble_tl
2793   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2794   \exp_args:No \@mkpream \g_@@_preamble_tl
2795   \@addtopreamble \@empty
2796   \endgroup
2797   \bool_if:NT \c_@@_recent_array_bool
2798   { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2799   \int_compare:nNnT { #1 } > \c_one_int
2800   {

```

```

2801 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2802 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2803 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2804 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2805 {
2806   {
2807     \int_if_zero:nTF \c@jCol
2808     { \int_eval:n { \c@iRow + 1 } }
2809     { \int_use:N \c@iRow }
2810   }
2811   { \int_eval:n { \c@jCol + 1 } }
2812   {
2813     \int_if_zero:nTF \c@jCol
2814     { \int_eval:n { \c@iRow + 1 } }
2815     { \int_use:N \c@iRow }
2816   }
2817   { \int_eval:n { \c@jCol + #1 } }
2818   { } % for the name of the block
2819 }
2820 }

```

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

```

2821 \RenewDocumentCommand \cellcolor { 0 { } m }
2822 {
2823   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2824   {
2825     \@@_rectanglecolor [ ##1 ]
2826     { \exp_not:n { ##2 } }
2827     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2828     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2829   }
2830   \ignorespaces
2831 }

```

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

```

2832 \cs_set_nopar:Npn \@sharp { #3 }
2833 \@arstrut
2834 \@preamble
2835 \null

```

We add some lines.

```

2836 \int_gadd:Nn \c@jCol { #1 - 1 }
2837 \int_compare:nNt \c@jCol > \g_@@_col_total_int
2838 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2839 \ignorespaces
2840 }

```

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`.

```

2841 \cs_new_protected:Npn \@_make_m_preamble:n #1
2842 {
2843   \str_case:nnF { #1 }
2844   {
2845     c { \@_make_m_preamble_i:n #1 }
2846     l { \@_make_m_preamble_i:n #1 }
2847     r { \@_make_m_preamble_i:n #1 }
2848     > { \@_make_m_preamble_ii:nn #1 }
2849     ! { \@_make_m_preamble_ii:nn #1 }
2850     @ { \@_make_m_preamble_ii:nn #1 }
2851     | { \@_make_m_preamble_iii:n #1 }
2852     p { \@_make_m_preamble_iv:nnn t #1 }

```

```

2853     m { \@@_make_m_preamble_iv:nnn c #1 }
2854     b { \@@_make_m_preamble_iv:nnn b #1 }
2855     w { \@@_make_m_preamble_v:nnnn { } #1 }
2856     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2857     \q_stop { }
2858   }
2859   {
2860     \cs_if_exist:cTF { NC @ find @ #1 }
2861     {
2862       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2863       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2864     }
2865     {
2866       \str_if_eq:nnTF { #1 } { S }
2867       { \@@_fatal:n { unknown~column~type~S } }
2868       { \@@_fatal:nn { unknown~column~type } { #1 } }
2869     }
2870   }
2871 }

```

For c, l and r

```

2872 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2873 {
2874   \tl_gput_right:Nn \g_@@_preamble_tl
2875   {
2876     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2877     #1
2878     < \@@_cell_end:
2879   }

```

We test for the presence of a <.

```

2880     \@@_make_m_preamble_x:n
2881   }

```

For >, ! and @

```

2882 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2883 {
2884   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2885   \@@_make_m_preamble:n
2886 }

```

For |

```

2887 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2888 {
2889   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2890   \@@_make_m_preamble:n
2891 }

```

For p, m and b

```

2892 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2893 {
2894   \tl_gput_right:Nn \g_@@_preamble_tl
2895   {
2896     > {
2897       \@@_cell_begin:
2898       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2899       \mode_leave_vertical:
2900       \arraybackslash
2901       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2902     }
2903     c
2904     < {
2905       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt

```

```

2906         \end { minipage }
2907     \@@_cell_end:
2908 }
2909 }

```

We test for the presence of a <.

```

2910     \@@_make_m_preamble_x:n
2911 }

```

For w and W

```

2912 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2913 {
2914     \tl_gput_right:Nn \g_@@_preamble_tl
2915     {
2916         > {
2917             \dim_set:Nn \l_@@_col_width_dim { #4 }
2918             \hbox_set:Nw \l_@@_cell_box
2919             \@@_cell_begin:
2920             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2921         }
2922         c
2923         < {
2924             \@@_cell_end:
2925             \hbox_set_end:
2926             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2927             #1
2928             \@@_adjust_size_box:
2929             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2930         }
2931     }

```

We test for the presence of a <.

```

2932     \@@_make_m_preamble_x:n
2933 }

```

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

```

2934 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2935 {
2936     \str_if_eq:nnTF { #1 } { < }
2937     \@@_make_m_preamble_ix:n
2938     { \@@_make_m_preamble:n { #1 } }
2939 }
2940 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2941 {
2942     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2943     \@@_make_m_preamble_x:n
2944 }

```

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).

```

2945 \cs_new_protected:Npn \@@_put_box_in_flow:
2946 {
2947     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2948     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2949     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2950     { \box_use_drop:N \l_tmpa_box }
2951     \@@_put_box_in_flow_i:
2952 }

```

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

```

2953 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2954 {
2955   \pgfpicture
2956     \@@_qpoint:n { row - 1 }
2957     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2958     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2959     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2960     \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).

```

2961     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2962     {
2963       \int_set:Nn \l_tmpa_int
2964       {
2965         \str_range:Nnn
2966           \l_@@_baseline_tl
2967           6
2968           { \tl_count:o \l_@@_baseline_tl }
2969       }
2970       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2971     }
2972     {
2973       \str_if_eq:eeTF \l_@@_baseline_tl { t }
2974       { \int_set_eq:NN \l_tmpa_int \c_one_int }
2975       {
2976         \str_if_eq:onTF \l_@@_baseline_tl { b }
2977         { \int_set_eq:NN \l_tmpa_int \c@iRow }
2978         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2979       }
2980       \bool_lazy_or:nnT
2981       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2982       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2983       {
2984         \@@_error:n { bad-value-for-baseline }
2985         \int_set_eq:NN \l_tmpa_int \c_one_int
2986       }
2987       \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2988     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2989   }
2990   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2991   \endpgfpicture
2992   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2993   \box_use_drop:N \l_tmpa_box
2994 }

```

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).

```

2995 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2996 {

```

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.

```

2997   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2998   {
2999     \int_compare:nNnT \c@jCol > \c_one_int

```

```

3000     {
3001         \box_set_wd:Nn \l_@@_the_array_box
3002         { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3003     }
3004 }

```

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

```

3005 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3006 \bool_if:NT \l_@@_caption_above_bool
3007 {
3008     \tl_if_empty:NF \l_@@_caption_tl
3009     {
3010         \bool_set_false:N \g_@@_caption_finished_bool
3011         \int_gzero:N \c@tabularnote
3012         \@@_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).

```

3013         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3014         {
3015             \tl_gput_right:Ne \g_@@_aux_tl
3016             {
3017                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3018                 { \int_use:N \g_@@_notes_caption_int }
3019             }
3020             \int_gzero:N \g_@@_notes_caption_int
3021         }
3022     }
3023 }

```

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

```

3024 \hbox
3025 {
3026     \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.

```

3027 \@@_create_extra_nodes:
3028 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3029 }

```

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

```

3030 \bool_lazy_any:nT
3031 {
3032     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3033     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3034     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3035 }
3036 \@@_insert_tabularnotes:
3037 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3038 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3039 \end { minipage }
3040 }

```

```

3041 \cs_new_protected:Npn \@@_insert_caption:
3042 {

```

```

3043 \tl_if_empty:NF \l_@@_caption_tl
3044 {
3045     \cs_if_exist:NTF \@capttype
3046     { \@@_insert_caption_i: }
3047     { \@@_error:n { caption~outside~float } }
3048 }
3049 }

```

```

3050 \cs_new_protected:Npn \@@_insert_caption_i:
3051 {
3052     \group_begin:

```

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

```

3053     \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.

```

3054 \IfPackageLoadedT { floatrow }
3055 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3056 \tl_if_empty:NTF \l_@@_short_caption_tl
3057 { \caption }
3058 { \caption [ \l_@@_short_caption_tl ] }
3059 { \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.

```

3060 \bool_if:NF \g_@@_caption_finished_bool
3061 {
3062     \bool_gset_true:N \g_@@_caption_finished_bool
3063     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3064     \int_gzero:N \c@tabularnote
3065 }
3066 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3067 \group_end:
3068 }

```

```

3069 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3070 {
3071     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3072     \@@_gredirect_none:n { tabularnote~below~the~tabular }
3073 }

```

```

3074 \cs_new_protected:Npn \@@_insert_tabularnotes:
3075 {
3076     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3077     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3078     \skip_vertical:N 0.65ex

```

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

```

3079 \group_begin:
3080 \l_@@_notes_code_before_tl
3081 \tl_if_empty:NF \g_@@_tabularnote_tl
3082 {
3083     \g_@@_tabularnote_tl \par
3084     \tl_gclear:N \g_@@_tabularnote_tl
3085 }

```

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.

```

3086 \int_compare:nNnT \c@tabularnote > \c_zero_int
3087 {
3088   \bool_if:NTF \l_@@_notes_para_bool
3089   {
3090     \begin { tabularnotes* }
3091     \seq_map_inline:Nn \g_@@_notes_seq
3092       { \@@_one_tabularnote:nn ##1 }
3093     \strut
3094     \end { tabularnotes* }

```

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

```

3095 \par
3096 }
3097 {
3098   \tabularnotes
3099   \seq_map_inline:Nn \g_@@_notes_seq
3100     { \@@_one_tabularnote:nn ##1 }
3101   \strut
3102   \endtabularnotes
3103 }
3104 }
3105 \unskip
3106 \group_end:
3107 \bool_if:NT \l_@@_notes_bottomrule_bool
3108 {
3109   \IfPackageLoadedTF { booktabs }
3110   {

```

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

```

3111 \skip_vertical:N \aboverulesep

```

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

```

3112 { \CT@arc@ \hrule height \heavyrulewidth }
3113 }
3114 { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3115 }
3116 \l_@@_notes_code_after_tl
3117 \seq_gclear:N \g_@@_notes_seq
3118 \seq_gclear:N \g_@@_notes_in_caption_seq
3119 \int_gzero:N \c@tabularnote
3120 }

```

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.

```

3121 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3122 {
3123   \tl_if_novalue:NTF { #1 }
3124   { \item }
3125   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3126 }

```

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`.

```

3127 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3128 {
3129   \pgfpicture
3130   \@@_qpoint:n { row - 1 }
3131   \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```



```

3132 \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3133 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3134 \endpgfpicture
3135 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3136 \int_if_zero:nT \l_@@_first_row_int
3137 {
3138   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3139   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3140 }
3141 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3142 }

```

Now, the general case.

```

3143 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3144 {

```

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

```

3145 \str_if_eq:eeT \l_@@_baseline_tl { t }
3146 { \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 .

```

3147 \pgfpicture
3148 \@@_qpoint:n { row - 1 }
3149 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3150 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3151 {
3152   \int_set:Nn \l_tmpa_int
3153   {
3154     \str_range:Nnn
3155     \l_@@_baseline_tl
3156     6
3157     { \tl_count:o \l_@@_baseline_tl }
3158   }
3159   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3160 }
3161 {
3162   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3163   \bool_lazy_or:nnT
3164   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3165   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3166   {
3167     \@@_error:n { bad~value~for~baseline }
3168     \int_set:Nn \l_tmpa_int 1
3169   }
3170   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3171 }
3172 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3173 \endpgfpicture
3174 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3175 \int_if_zero:nT \l_@@_first_row_int
3176 {
3177   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3178   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3179 }
3180 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3181 }

```

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.

```

3182 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3183 {

```

We will compute the real width of both delimiters used.

```

3184 \dim_zero_new:N \l_@@_real_left_delim_dim
3185 \dim_zero_new:N \l_@@_real_right_delim_dim
3186 \hbox_set:Nn \l_tmpb_box
3187 {
3188   \m@th % added 2024/11/21
3189   \c_math_toggle_token
3190   \left #1
3191   \vcenter
3192   {
3193     \vbox_to_ht:nn
3194     { \box_ht_plus_dp:N \l_tmpa_box }
3195     { }
3196   }
3197   \right .
3198   \c_math_toggle_token
3199 }
3200 \dim_set:Nn \l_@@_real_left_delim_dim
3201 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3202 \hbox_set:Nn \l_tmpb_box
3203 {
3204   \m@th % added 2024/11/21
3205   \c_math_toggle_token
3206   \left .
3207   \vbox_to_ht:nn
3208   { \box_ht_plus_dp:N \l_tmpa_box }
3209   { }
3210   \right #2
3211   \c_math_toggle_token
3212 }
3213 \dim_set:Nn \l_@@_real_right_delim_dim
3214 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3215 \skip_horizontal:N \l_@@_left_delim_dim
3216 \skip_horizontal:N -\l_@@_real_left_delim_dim
3217 @@_put_box_in_flow:
3218 \skip_horizontal:N \l_@@_right_delim_dim
3219 \skip_horizontal:N -\l_@@_real_right_delim_dim
3220 }

```

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).

```

3221 \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).

```

3222 {
3223   \peek_remove_spaces:n
3224   {
3225     \peek_meaning:NTF \end
3226     \@@_analyze_end:Nn
3227     {
3228       \@@_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`).

```

3229 \@@_array:o \g_@@_array_preamble_tl
3230 }

```

```

3231     }
3232   }
3233   {
3234     \@@_create_col_nodes:
3235     \endarray
3236   }

```

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

```

3237 \NewDocumentEnvironment { @@-light-syntax } { b }
3238 {

```

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`.

```

3239   \tl_if_empty:nT { #1 }
3240   { \@@_fatal:n { empty-environment } }
3241   \tl_if_in:nnT { #1 } { & }
3242   { \@@_fatal:n { ampersand-in-light-syntax } }
3243   \tl_if_in:nnT { #1 } { \ }
3244   { \@@_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`.

```

3245   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3246   }

```

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.

```

3247   {
3248     \@@_create_col_nodes:
3249     \endarray
3250   }

3251 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3252 {
3253   \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).

```

3254   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3255   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3256   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3257     \seq_set_split:Nee
3258     \seq_set_split:Non
3259     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3260   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3261   \tl_if_empty:NF \l_tmpa_tl
3262   { \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.

```

3263   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3264   { \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`).

```
3265 \tl_build_begin:N \l_@@_new_body_tl
3266 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3267 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3268 \@@_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).

```
3269 \seq_map_inline:Nn \l_@@_rows_seq
3270 {
3271   \tl_build_put_right:Nn \l_@@_new_body_tl { \ }
3272   \@@_line_with_light_syntax:n { ##1 }
3273 }
3274 \tl_build_end:N \l_@@_new_body_tl
3275 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3276 {
3277   \int_set:Nn \l_@@_last_col_int
3278     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3279 }
```

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.

```
3280 \@@_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`).

```
3281 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3282 }
3283 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3284 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3285 {
3286   \seq_clear_new:N \l_@@_cells_seq
3287   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3288   \int_set:Nn \l_@@_nb_cols_int
3289   {
3290     \int_max:nn
3291       \l_@@_nb_cols_int
3292       { \seq_count:N \l_@@_cells_seq }
3293   }
3294   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3295   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3296   \seq_map_inline:Nn \l_@@_cells_seq
3297     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3298 }
```

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`.

```
3299 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3300 {
3301   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3302   { \@@_fatal:n { empty-environment } }
```

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

```
3303 \end { #2 }
3304 }
```

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`).

```

3305 \cs_new:Npn \@@_create_col_nodes:
3306 {
3307   \crrc
3308   \int_if_zero:nT \l_@@_first_col_int
3309   {
3310     \omit
3311     \hbox_overlap_left:n
3312     {
3313       \bool_if:NT \l_@@_code_before_bool
3314       { \pgfsys@markposition { \@@_env: - col - 0 } }
3315       \pgfpicture
3316       \pgfrememberpicturepositiononpagetrue
3317       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3318       \str_if_empty:NF \l_@@_name_str
3319       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3320       \endpgfpicture
3321       \skip_horizontal:N 2\col@sep
3322       \skip_horizontal:N \g_@@_width_first_col_dim
3323     }
3324     &
3325   }
3326   \omit

```

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

```

3327   \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`).

```

3328   \int_if_zero:nTF \l_@@_first_col_int
3329   {
3330     \bool_if:NT \l_@@_code_before_bool
3331     {
3332       \hbox
3333       {
3334         \skip_horizontal:N -0.5\arrayrulewidth
3335         \pgfsys@markposition { \@@_env: - col - 1 }
3336         \skip_horizontal:N 0.5\arrayrulewidth
3337       }
3338     }
3339     \pgfpicture
3340     \pgfrememberpicturepositiononpagetrue
3341     \pgfcoordinate { \@@_env: - col - 1 }
3342     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3343     \str_if_empty:NF \l_@@_name_str
3344     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3345     \endpgfpicture
3346   }
3347   {
3348     \bool_if:NT \l_@@_code_before_bool
3349     {
3350       \hbox
3351       {
3352         \skip_horizontal:N 0.5\arrayrulewidth
3353         \pgfsys@markposition { \@@_env: - col - 1 }
3354         \skip_horizontal:N -0.5\arrayrulewidth
3355       }
3356     }
3357     \pgfpicture
3358     \pgfrememberpicturepositiononpagetrue
3359     \pgfcoordinate { \@@_env: - col - 1 }
3360     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }

```

```

3361 \str_if_empty:NF \l_@@_name_str
3362 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3363 \endpgfpicture
3364 }

```

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.

```

3365 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3366 \bool_if:NF \l_@@_auto_columns_width_bool
3367 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3368 {
3369   \bool_lazy_and:nnTF
3370     \l_@@_auto_columns_width_bool
3371     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3372     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3373     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3374     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3375 }
3376 \skip_horizontal:N \g_tmpa_skip
3377 \hbox
3378 {
3379   \bool_if:NT \l_@@_code_before_bool
3380   {
3381     \hbox
3382     {
3383       \skip_horizontal:N -0.5\arrayrulewidth
3384       \pgfsys@markposition { \@@_env: - col - 2 }
3385       \skip_horizontal:N 0.5\arrayrulewidth
3386     }
3387   }
3388   \pgfpicture
3389   \pgfrememberpicturepositiononpagetrue
3390   \pgfcoordinate { \@@_env: - col - 2 }
3391   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3392   \str_if_empty:NF \l_@@_name_str
3393   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3394   \endpgfpicture
3395 }

```

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.

```

3396 \int_gset_eq:NN \g_tmpa_int \c_one_int
3397 \bool_if:NTF \g_@@_last_col_found_bool
3398 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3399 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3400 {
3401   &
3402   \omit
3403   \int_gincr:N \g_tmpa_int

```

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

```

3404 \skip_horizontal:N \g_tmpa_skip
3405 \bool_if:NT \l_@@_code_before_bool
3406 {
3407   \hbox
3408   {
3409     \skip_horizontal:N -0.5\arrayrulewidth
3410     \pgfsys@markposition
3411     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3412     \skip_horizontal:N 0.5\arrayrulewidth

```

```

3413     }
3414 }

```

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

```

3415 \pgfpicture
3416 \pgfrememberpicturepositiononpagetrue
3417 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3418 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3419 \str_if_empty:NF \l_@@_name_str
3420 {
3421 \pgfnodealias
3422 { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3423 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3424 }
3425 \endpgfpicture
3426 }

3427 &
3428 \omit

```

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

```

3429 \int_if_zero:nT \g_@@_col_total_int
3430 { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3431 \skip_horizontal:N \g_tmpa_skip
3432 \int_gincr:N \g_tmpa_int
3433 \bool_lazy_any:nF
3434 {
3435 \g_@@_delims_bool
3436 \l_@@_tabular_bool
3437 { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3438 \l_@@_exterior_arraycolsep_bool
3439 \l_@@_bar_at_end_of_pream_bool
3440 }
3441 { \skip_horizontal:N -\col@sep }
3442 \bool_if:NT \l_@@_code_before_bool
3443 {
3444 \hbox
3445 {
3446 \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.

```

3447 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3448 { \skip_horizontal:N -\arraycolsep }
3449 \pgfsys@markposition
3450 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3451 \skip_horizontal:N 0.5\arrayrulewidth
3452 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3453 { \skip_horizontal:N \arraycolsep }
3454 }
3455 }
3456 \pgfpicture
3457 \pgfrememberpicturepositiononpagetrue
3458 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3459 {
3460 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3461 {
3462 \pgfpoint
3463 { - 0.5 \arrayrulewidth - \arraycolsep }
3464 \c_zero_dim
3465 }
3466 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }

```

```

3467     }
3468     \str_if_empty:NF \l_@@_name_str
3469     {
3470         \pgfnodealias
3471         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3472         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3473     }
3474     \endpgfpicture

3475     \bool_if:NT \g_@@_last_col_found_bool
3476     {
3477         \hbox_overlap_right:n
3478         {
3479             \skip_horizontal:N \g_@@_width_last_col_dim
3480             \skip_horizontal:N \col@sep
3481             \bool_if:NT \l_@@_code_before_bool
3482             {
3483                 \pgfsys@markposition
3484                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3485             }
3486             \pgfpicture
3487             \pgfrememberpicturerepositiononpagetrue
3488             \pgfcoordinate
3489             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3490             \pgfpintorigin
3491             \str_if_empty:NF \l_@@_name_str
3492             {
3493                 \pgfnodealias
3494                 {
3495                     \l_@@_name_str - col
3496                     - \int_eval:n { \g_@@_col_total_int + 1 }
3497                 }
3498                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3499             }
3500             \endpgfpicture
3501         }
3502     }
3503     % \cr
3504 }

```

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

```

3505 \tl_const:Nn \c_@@_preamble_first_col_tl
3506 {
3507     >
3508     {

```

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

```

3509         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3510         \bool_gset_true:N \g_@@_after_col_zero_bool
3511         \@@_begin_of_row:
3512         \hbox_set:Nw \l_@@_cell_box
3513         \@@_math_toggle:
3514         \@@_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”.

```

3515         \int_compare:nNnT \c@iRow > \c_zero_int
3516         {
3517             \bool_lazy_or:nnT
3518             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3519             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }

```



```

3520         {
3521             \l_@@_code_for_first_col_tl
3522             \xglobal \colorlet { nicematrix-first-col } { . }
3523         }
3524     }
3525 }

```

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`.

```

3526     l
3527     <
3528     {
3529         \@@_math_toggle:
3530         \hbox_set_end:
3531         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3532         \@@_adjust_size_box:
3533         \@@_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.

```

3534         \dim_gset:Nn \g_@@_width_first_col_dim
3535         { \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.

```

3536         \hbox_overlap_left:n
3537         {
3538             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3539             \@@_node_for_cell:
3540             { \box_use_drop:N \l_@@_cell_box }
3541             \skip_horizontal:N \l_@@_left_delim_dim
3542             \skip_horizontal:N \l_@@_left_margin_dim
3543             \skip_horizontal:N \l_@@_extra_left_margin_dim
3544         }
3545         \bool_gset_false:N \g_@@_empty_cell_bool
3546         \skip_horizontal:N -2\col@sep
3547     }
3548 }

```

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

```

3549 \tl_const:Nn \c_@@_preamble_last_col_tl
3550 {
3551     >
3552     {
3553         \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).

```

3554         \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.

```

3555         \bool_gset_true:N \g_@@_last_col_found_bool
3556         \int_gincr:N \c@jCol
3557         \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3558         \hbox_set:Nw \l_@@_cell_box
3559         \@@_math_toggle:
3560         \@@_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”.

```

3561         \int_compare:nNnT \c@iRow > \c_zero_int
3562         {
3563             \bool_lazy_or:nnT
3564             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3565             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3566             {
3567                 \l_@@_code_for_last_col_tl

```

```

3568         \xglobal \colorlet { nicematrix-last-col } { . }
3569     }
3570 }
3571 }
3572 1
3573 <
3574 {
3575     \@@_math_toggle:
3576     \hbox_set_end:
3577     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3578     \@@_adjust_size_box:
3579     \@@_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.

```

3580 \dim_gset:Nn \g_@@_width_last_col_dim
3581 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3582 \skip_horizontal:N -2\col@sep

```

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

```

3583 \hbox_overlap_right:n
3584 {
3585     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3586     {
3587         \skip_horizontal:N \l_@@_right_delim_dim
3588         \skip_horizontal:N \l_@@_right_margin_dim
3589         \skip_horizontal:N \l_@@_extra_right_margin_dim
3590         \@@_node_for_cell:
3591     }
3592 }
3593 \bool_gset_false:N \g_@@_empty_cell_bool
3594 }
3595 }

```

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

```

3596 \NewDocumentEnvironment { NiceArray } { }
3597 {
3598     \bool_gset_false:N \g_@@_delims_bool
3599     \str_if_empty:NT \g_@@_name_env_str
3600     { \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).

```

3601 \NiceArrayWithDelims . .
3602 }
3603 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3604 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3605 {
3606     \NewDocumentEnvironment { #1 NiceArray } { }
3607     {
3608         \bool_gset_true:N \g_@@_delims_bool
3609         \str_if_empty:NT \g_@@_name_env_str
3610         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3611         \@@_test_if_math_mode:
3612         \NiceArrayWithDelims #2 #3
3613     }
3614     { \endNiceArrayWithDelims }
3615 }

```

```

3616 \@@_def_env:nnn p ( )
3617 \@@_def_env:nnn b [ ]
3618 \@@_def_env:nnn B \{ \}
3619 \@@_def_env:nnn v | |
3620 \@@_def_env:nnn V \l \l

```

13 The environment {NiceMatrix} and its variants

```

3621 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3622 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3623 {
3624   \bool_set_false:N \l_@@_preamble_bool
3625   \tl_clear:N \l_tmpa_tl
3626   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3627     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3628   \tl_put_right:Nn \l_tmpa_tl
3629     {
3630       *
3631       {
3632         \int_case:nnF \l_@@_last_col_int
3633         {
3634           { -2 } { \c@MaxMatrixCols }
3635           { -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).

```

3636       }
3637       { \int_eval:n { \l_@@_last_col_int - 1 } }
3638     }
3639     { #2 }
3640   }
3641   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3642   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3643 }
3644 \clist_map_inline:nn { p , b , B , v , V }
3645 {
3646   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3647   {
3648     \bool_gset_true:N \g_@@_delims_bool
3649     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3650     \int_if_zero:nT \l_@@_last_col_int
3651     {
3652       \bool_set_true:N \l_@@_last_col_without_value_bool
3653       \int_set:Nn \l_@@_last_col_int { -1 }
3654     }
3655     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3656     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3657   }
3658   { \use:c { end #1 NiceArray } }
3659 }

```

We define also an environment {NiceMatrix}

```

3660 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3661 {
3662   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3663   \int_if_zero:nT \l_@@_last_col_int
3664   {
3665     \bool_set_true:N \l_@@_last_col_without_value_bool
3666     \int_set:Nn \l_@@_last_col_int { -1 }
3667   }

```

```

3668 \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3669 \bool_lazy_or:nnT
3670 { \clist_if_empty_p:N \l_@@_vlines_clist }
3671 { \l_@@_except_borders_bool }
3672 { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3673 \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3674 }
3675 { \endNiceArray }

```

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

```

3676 \cs_new_protected:Npn \@@_NotEmpty:
3677 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3678 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3679 {

```

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`.

```

3680 \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3681 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3682 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3683 \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3684 \tl_if_empty:NF \l_@@_short_caption_tl
3685 {
3686 \tl_if_empty:NT \l_@@_caption_tl
3687 {
3688 \@@_error_or_warning:n { short-caption-without~caption }
3689 \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3690 }
3691 }
3692 \tl_if_empty:NF \l_@@_label_tl
3693 {
3694 \tl_if_empty:NT \l_@@_caption_tl
3695 { \@@_error_or_warning:n { label~without~caption } }
3696 }
3697 \NewDocumentEnvironment { TabularNote } { b }
3698 {
3699 \bool_if:NTF \l_@@_in_code_after_bool
3700 { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3701 {
3702 \tl_if_empty:NF \g_@@_tabularnote_tl
3703 { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3704 \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3705 }
3706 }
3707 { }
3708 \@@_settings_for_tabular:
3709 \NiceArray { #2 }
3710 }
3711 { \endNiceArray }
3712 \cs_new_protected:Npn \@@_settings_for_tabular:
3713 {
3714 \bool_set_true:N \l_@@_tabular_bool
3715 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3716 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3717 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3718 }

```

```

3719 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }

```

```

3720 {
3721   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3722   \dim_zero_new:N \l_@@_width_dim
3723   \dim_set:Nn \l_@@_width_dim { #1 }
3724   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3725   \@@_settings_for_tabular:
3726   \NiceArray { #3 }
3727 }
3728 {
3729   \endNiceArray
3730   \int_if_zero:nT \g_@@_total_X_weight_int
3731   { \@@_error:n { NiceTabularX~without~X } }
3732 }

3733 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3734 {
3735   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3736   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3737   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3738   \@@_settings_for_tabular:
3739   \NiceArray { #3 }
3740 }
3741 { \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`).

```

3742 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3743 {
3744   \bool_lazy_all:nT
3745   {
3746     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3747     \l_@@_hvlines_bool
3748     { ! \g_@@_delims_bool }
3749     { ! \l_@@_except_borders_bool }
3750   }
3751   {
3752     \bool_set_true:N \l_@@_except_borders_bool
3753     \clist_if_empty:NF \l_@@_corners_clist
3754     { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3755     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3756     {
3757       \@@_stroke_block:nnn
3758       {
3759         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3760         draw = \l_@@_rules_color_tl
3761       }
3762       { 1-1 }
3763       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3764     }
3765   }
3766 }

3767 \cs_new_protected:Npn \@@_after_array:
3768 {

```

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`.

```
3769 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3770 \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.

```
3771 \bool_if:NT \g_@@_last_col_found_bool
3772 { \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`.

```
3773 \bool_if:NT \l_@@_last_col_without_value_bool
3774 { \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.

```
3775 \bool_if:NT \l_@@_last_row_without_value_bool
3776 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3777 \tl_gput_right:Ne \g_@@_aux_tl
3778 {
3779   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3780   {
3781     \int_use:N \l_@@_first_row_int ,
3782     \int_use:N \c@iRow ,
3783     \int_use:N \g_@@_row_total_int ,
3784     \int_use:N \l_@@_first_col_int ,
3785     \int_use:N \c@jCol ,
3786     \int_use:N \g_@@_col_total_int
3787   }
3788 }
```

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`).

```
3789 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3790 {
3791   \tl_gput_right:Ne \g_@@_aux_tl
3792   {
3793     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3794     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3795   }
3796 }
3797 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3798 {
3799   \tl_gput_right:Ne \g_@@_aux_tl
3800   {
3801     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3802     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3803     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3804     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3805   }
3806 }
```

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

```
3807 \@@_create_diag_nodes:
```

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

```

3808 \pgfpicture
3809 \int_step_inline:nn \c@iRow
3810 {
3811   \pgfnodealias
3812   { \@@_env: - ##1 - last }
3813   { \@@_env: - ##1 - \int_use:N \c@jCol }
3814 }
3815 \int_step_inline:nn \c@jCol
3816 {
3817   \pgfnodealias
3818   { \@@_env: - last - ##1 }
3819   { \@@_env: - \int_use:N \c@iRow - ##1 }
3820 }
3821 \str_if_empty:NF \l_@@_name_str
3822 {
3823   \int_step_inline:nn \c@iRow
3824   {
3825     \pgfnodealias
3826     { \l_@@_name_str - ##1 - last }
3827     { \@@_env: - ##1 - \int_use:N \c@jCol }
3828   }
3829   \int_step_inline:nn \c@jCol
3830   {
3831     \pgfnodealias
3832     { \l_@@_name_str - last - ##1 }
3833     { \@@_env: - \int_use:N \c@iRow - ##1 }
3834   }
3835 }
3836 \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.

```

3837 \bool_if:NT \l_@@_parallelize_diags_bool
3838 {
3839   \int_gzero_new:N \g_@@_ddots_int
3840   \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.

```

3841   \dim_gzero_new:N \g_@@_delta_x_one_dim
3842   \dim_gzero_new:N \g_@@_delta_y_one_dim
3843   \dim_gzero_new:N \g_@@_delta_x_two_dim
3844   \dim_gzero_new:N \g_@@_delta_y_two_dim
3845 }
3846 \int_zero_new:N \l_@@_initial_i_int
3847 \int_zero_new:N \l_@@_initial_j_int
3848 \int_zero_new:N \l_@@_final_i_int
3849 \int_zero_new:N \l_@@_final_j_int
3850 \bool_set_false:N \l_@@_initial_open_bool
3851 \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.

```

3852 \bool_if:NT \l_@@_small_bool
3853 {

```

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

```

3854 \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3855 \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.

```

3856 \dim_set:Nn \l_@@_xdots_shorten_start_dim
3857 { 0.6 \l_@@_xdots_shorten_start_dim }
3858 \dim_set:Nn \l_@@_xdots_shorten_end_dim
3859 { 0.6 \l_@@_xdots_shorten_end_dim }
3860 }

```

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

```

3861 \@@_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.

```

3862 \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-*}`).

```

3863 \@@_adjust_pos_of_blocks_seq:
3864 \@@_deal_with_rounded_corners:
3865 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3866 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3867 \IfPackageLoadedT { tikz }
3868 {
3869   \tikzset
3870   {
3871     every-picture / .style =
3872     {
3873       overlay ,
3874       remember~picture ,
3875       name~prefix = \@@_env: -
3876     }
3877   }
3878 }
3879 \bool_if:NT \c_@@_recent_array_bool
3880 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3881 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3882 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3883 \cs_set_eq:NN \OverBrace \@@_OverBrace
3884 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3885 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3886 \cs_set_eq:NN \line \@@_line
3887 \g_@@_pre_code_after_tl
3888 \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.

```

3889 \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).

```

3890 \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`).

```
3891 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3892 { \@@_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:`.

```
3893 \bool_set_true:N \l_@@_in_code_after_bool
3894 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3895 \scan_stop:
3896 \tl_gclear:N \g_nicematrix_code_after_tl
3897 \group_end:
```

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

```
3898 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3899 \tl_if_empty:NF \g_@@_pre_code_before_tl
3900 {
3901   \tl_gput_right:Ne \g_@@_aux_tl
3902   {
3903     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3904     { \exp_not:o \g_@@_pre_code_before_tl }
3905   }
3906   \tl_gclear:N \g_@@_pre_code_before_tl
3907 }
3908 \tl_if_empty:NF \g_nicematrix_code_before_tl
3909 {
3910   \tl_gput_right:Ne \g_@@_aux_tl
3911   {
3912     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3913     { \exp_not:o \g_nicematrix_code_before_tl }
3914   }
3915   \tl_gclear:N \g_nicematrix_code_before_tl
3916 }

3917 \str_gclear:N \g_@@_name_env_str
3918 \@@_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`.

```
3919 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3920 }
```

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`.

```
3921 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3922 { \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

¹²e.g. `\color[rgb]{0.5,0.5,0}`

`\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).

```

3923 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3924 {
3925   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3926   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3927 }

```

The following command must *not* be protected.

```

3928 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3929 {
3930   { #1 }
3931   { #2 }
3932   {
3933     \int_compare:nNnTF { #3 } > { 98 }
3934     { \int_use:N \c@iRow }
3935     { #3 }
3936   }
3937   {
3938     \int_compare:nNnTF { #4 } > { 98 }
3939     { \int_use:N \c@jCol }
3940     { #4 }
3941   }
3942   { #5 }
3943 }

```

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).

```

3944 \hook_gput_code:nnn { begindocument } { . }
3945 {
3946   \cs_new_protected:Npe \@@_draw_dotted_lines:
3947   {
3948     \c_@@_pgfortikzpicture_tl
3949     \@@_draw_dotted_lines_i:
3950     \c_@@_endpgfortikzpicture_tl
3951   }
3952 }

```

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

```

3953 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3954 {
3955   \pgfrememberpicturepositiononpagetrue
3956   \pgf@relevantforpicturesizefalse
3957   \g_@@_HVdotsfor_lines_tl
3958   \g_@@_Vdots_lines_tl
3959   \g_@@_Ddots_lines_tl
3960   \g_@@_Iddots_lines_tl
3961   \g_@@_Cdots_lines_tl
3962   \g_@@_Ldots_lines_tl
3963 }

3964 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3965 {
3966   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3967   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3968 }

```

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

```

3969 \pgfdeclareshape { @@_diag_node }
3970 {
3971   \savedanchor { \five }
3972   {
3973     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3974     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3975   }
3976   \anchor { 5 } { \five }
3977   \anchor { center } { \pgfpointorigin }
3978   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3979   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3980   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3981   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3982   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
3983   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3984   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3985   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3986   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3987   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3988 }

```

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).

```

3989 \cs_new_protected:Npn \@@_create_diag_nodes:
3990 {
3991   \pgfpicture
3992   \pgfrememberpicturepositiononpagetrue
3993   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3994   {
3995     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3996     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3997     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3998     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3999     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4000     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4001     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4002     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4003     \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.

```

4004     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4005     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4006     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4007     \str_if_empty:NF \l_@@_name_str
4008     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4009   }

```

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

```

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

```

```

4023         { \@@_env: - \int_use:N \l_tmpa_int }
4024     \pgfnodealias
4025         { \l_@@_name_str - last }
4026         { \@@_env: - last }
4027     }
4028 \endpgfpicture
4029 }

```

16 We draw the dotted lines

A dotted line will be said *open* in one of its extremities when it stops on the edge of the matrix and *closed* otherwise. In the following matrix, the dotted line is closed on its left extremity and open on its right.

$$\begin{pmatrix} a+b+c & a+b & a \\ a & \cdots & \cdots \\ a & a+b & a+b+c \end{pmatrix}$$

The command `\@@_find_extremities_of_line:nnnn` takes four arguments:

- the first argument is the row of the cell where the command was issued;
- the second argument is the column of the cell where the command was issued;
- the third argument is the x -value of the orientation vector of the line;
- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

```

4030 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4031 {

```

First, we declare the current cell as “dotted” because we forbid intersections of dotted lines.

```

4032     \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4033     \int_set:Nn \l_@@_initial_i_int { #1 }
4034     \int_set:Nn \l_@@_initial_j_int { #2 }
4035     \int_set:Nn \l_@@_final_i_int { #1 }
4036     \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```

4037     \bool_set_false:N \l_@@_stop_loop_bool
4038     \bool_do_until:Nn \l_@@_stop_loop_bool
4039     {
4040         \int_add:Nn \l_@@_final_i_int { #3 }
4041         \int_add:Nn \l_@@_final_j_int { #4 }
4042         \bool_set_false:N \l_@@_final_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4043     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4044     \if_int_compare:w #3 = \c_one_int
4045     \bool_set_true:N \l_@@_final_open_bool
4046   \else:
4047     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4048     \bool_set_true:N \l_@@_final_open_bool
4049   \fi:
4050 \fi:
4051 \else:
4052   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4053   \if_int_compare:w #4 = -1
4054   \bool_set_true:N \l_@@_final_open_bool
4055   \fi:
4056 \else:
4057   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4058   \if_int_compare:w #4 = \c_one_int
4059   \bool_set_true:N \l_@@_final_open_bool
4060   \fi:
4061 \fi:
4062 \fi:
4063 \fi:
4064 \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.

```

4065   {

```

We do a step backwards.

```

4066     \int_sub:Nn \l_@@_final_i_int { #3 }
4067     \int_sub:Nn \l_@@_final_j_int { #4 }
4068     \bool_set_true:N \l_@@_stop_loop_bool
4069   }

```

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`.

```

4070   {
4071     \cs_if_exist:cTF
4072     {
4073       @@ _ dotted _
4074       \int_use:N \l_@@_final_i_int -
4075       \int_use:N \l_@@_final_j_int
4076     }
4077     {
4078       \int_sub:Nn \l_@@_final_i_int { #3 }
4079       \int_sub:Nn \l_@@_final_j_int { #4 }
4080       \bool_set_true:N \l_@@_final_open_bool
4081       \bool_set_true:N \l_@@_stop_loop_bool
4082     }
4083   }
4084   \cs_if_exist:cTF
4085   {
4086     pgf @ sh @ ns @ \@@_env:
4087     - \int_use:N \l_@@_final_i_int
4088     - \int_use:N \l_@@_final_j_int
4089   }
4090   { \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.

```

4091   {

```

```

4092         \cs_set_nopar:cpn
4093         {
4094             @@ _ dotted _
4095             \int_use:N \l_@@_final_i_int -
4096             \int_use:N \l_@@_final_j_int
4097         }
4098         { }
4099     }
4100 }
4101 }
4102 }

```

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

```

4103     \bool_set_false:N \l_@@_stop_loop_bool

```

The following line of code is only for efficiency in the following loop.

```

4104     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4105     \bool_do_until:Nn \l_@@_stop_loop_bool
4106     {
4107         \int_sub:Nn \l_@@_initial_i_int { #3 }
4108         \int_sub:Nn \l_@@_initial_j_int { #4 }
4109         \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.

```

4110         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4111         \if_int_compare:w #3 = \c_one_int
4112             \bool_set_true:N \l_@@_initial_open_bool
4113         \else:

```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```

4114         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4115             \bool_set_true:N \l_@@_initial_open_bool
4116         \fi:
4117     \fi:
4118 \else:
4119     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4120         \if_int_compare:w #4 = \c_one_int
4121             \bool_set_true:N \l_@@_initial_open_bool
4122         \fi:
4123     \else:
4124         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4125             \if_int_compare:w #4 = -1
4126                 \bool_set_true:N \l_@@_initial_open_bool
4127             \fi:
4128         \fi:
4129     \fi:
4130 \fi:
4131 \bool_if:NTF \l_@@_initial_open_bool
4132 {
4133     \int_add:Nn \l_@@_initial_i_int { #3 }
4134     \int_add:Nn \l_@@_initial_j_int { #4 }
4135     \bool_set_true:N \l_@@_stop_loop_bool
4136 }
4137 {
4138     \cs_if_exist:cTF
4139     {
4140         @@ _ dotted _
4141         \int_use:N \l_@@_initial_i_int -
4142         \int_use:N \l_@@_initial_j_int
4143     }

```

```

4144     {
4145         \int_add:Nn \l_@@_initial_i_int { #3 }
4146         \int_add:Nn \l_@@_initial_j_int { #4 }
4147         \bool_set_true:N \l_@@_initial_open_bool
4148         \bool_set_true:N \l_@@_stop_loop_bool
4149     }
4150     {
4151         \cs_if_exist:cTF
4152         {
4153             pgf @ sh @ ns @ \@@_env:
4154             - \int_use:N \l_@@_initial_i_int
4155             - \int_use:N \l_@@_initial_j_int
4156         }
4157         { \bool_set_true:N \l_@@_stop_loop_bool }
4158         {
4159             \cs_set_nopar:cpn
4160             {
4161                 @@ _ dotted _
4162                 \int_use:N \l_@@_initial_i_int -
4163                 \int_use:N \l_@@_initial_j_int
4164             }
4165             { }
4166         }
4167     }
4168 }
4169 }

```

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.

```

4170 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4171 {
4172     { \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`.

```

4173     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4174     { \int_use:N \l_@@_final_i_int }
4175     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4176     { } % for the name of the block
4177 }
4178 }

```

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 known wheter 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.

```

4179 \cs_new_protected:Npn \@@_open_shorten:
4180 {
4181     \bool_if:NT \l_@@_initial_open_bool
4182     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4183     \bool_if:NT \l_@@_final_open_bool
4184     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4185 }

```

The following commmand (*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).

```

4186 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4187 {
4188     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4189 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4190 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4191 \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`.

```

4192 \seq_if_empty:NF \g_@@_submatrix_seq
4193 {
4194   \seq_map_inline:Nn \g_@@_submatrix_seq
4195     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4196 }
4197 }

```

#1 and **#2** are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. **#3**, **#4**, **#5** and **#6** are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4198 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4199 {
4200   \if_int_compare:w #3 > #1
4201   \else:
4202     \if_int_compare:w #1 > #5
4203     \else:
4204       \if_int_compare:w #4 > #2
4205       \else:
4206         \if_int_compare:w #2 > #6
4207         \else:
4208           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4209           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4210           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4211           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4212         \fi:
4213       \fi:
4214     \fi:
4215   \fi:
4216 }

4217 \cs_new_protected:Npn \@@_set_initial_coords:
4218 {
4219   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4220   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4221 }
4222 \cs_new_protected:Npn \@@_set_final_coords:
4223 {

```



```

4224 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4225 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4226 }
4227 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4228 {
4229 \pgfpointanchor
4230 {
4231 \@@_env:
4232 - \int_use:N \l_@@_initial_i_int
4233 - \int_use:N \l_@@_initial_j_int
4234 }
4235 { #1 }
4236 \@@_set_initial_coords:
4237 }
4238 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4239 {
4240 \pgfpointanchor
4241 {
4242 \@@_env:
4243 - \int_use:N \l_@@_final_i_int
4244 - \int_use:N \l_@@_final_j_int
4245 }
4246 { #1 }
4247 \@@_set_final_coords:
4248 }
4249 \cs_new_protected:Npn \@@_open_x_initial_dim:
4250 {
4251 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4252 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4253 {
4254 \cs_if_exist:cT
4255 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4256 {
4257 \pgfpointanchor
4258 { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4259 { west }
4260 \dim_set:Nn \l_@@_x_initial_dim
4261 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4262 }
4263 }

```

If, in fact, all the cells of the column are empty (no PGF/Tikz nodes in those cells).

```

4264 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4265 {
4266 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4267 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4268 \dim_add:Nn \l_@@_x_initial_dim \col@sep
4269 }
4270 }
4271 \cs_new_protected:Npn \@@_open_x_final_dim:
4272 {
4273 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4274 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4275 {
4276 \cs_if_exist:cT
4277 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4278 {
4279 \pgfpointanchor
4280 { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4281 { east }
4282 \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4283 { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4284 }

```

```
4285     }
```

If, in fact, all the cells of the columns are empty (no PGF/Tikz nodes in those cells).

```
4286     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4287     {
4288         \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4289         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4290         \dim_sub:Nn \l_@@_x_final_dim \col@sep
4291     }
4292 }
```

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.

```
4293 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4294 {
4295     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4296     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4297     {
4298         \@@_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.

```
4299         \group_begin:
4300         \@@_open_shorten:
4301         \int_if_zero:nTF { #1 }
4302         { \color { nicematrix-first-row } }
4303         {
```

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.

```
4304             \int_compare:nNnT { #1 } = \l_@@_last_row_int
4305             { \color { nicematrix-last-row } }
4306         }
4307         \keys_set:nn { nicematrix / xdots } { #3 }
4308         \@@_color:o \l_@@_xdots_color_tl
4309         \@@_actually_draw_Ldots:
4310     \group_end:
4311 }
4312 }
```

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`.

```
4313 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4314 {
4315     \bool_if:NTF \l_@@_initial_open_bool
4316     {
4317         \@@_open_x_initial_dim:
4318         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4319         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4320     }
4321     { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4322 \bool_if:NTF \l_@@_final_open_bool
4323 {
4324   \@@_open_x_final_dim:
4325   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4326   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4327 }
4328 { \@@_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.

```

4329 \bool_lazy_all:nTF
4330 {
4331   \l_@@_initial_open_bool
4332   \l_@@_final_open_bool
4333   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4334 }
4335 {
4336   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4337   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4338 }

```

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 (?).

```

4339 {
4340   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4341   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4342 }
4343 \@@_draw_line:
4344 }

```

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.

```

4345 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4346 {
4347   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4348   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4349   {
4350     \@@_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.

```

4351 \group_begin:
4352 \@@_open_shorten:
4353 \int_if_zero:nTF { #1 }
4354 { \color { nicematrix-first-row } }
4355 {

```

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.

```

4356 \int_compare:nNnT { #1 } = \l_@@_last_row_int
4357 { \color { nicematrix-last-row } }
4358 }
4359 \keys_set:nn { nicematrix / xdots } { #3 }
4360 \@@_color:o \l_@@_xdots_color_tl
4361 \@@_actually_draw_Cdots:
4362 \group_end:
4363 }
4364 }

```

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.

```

4365 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4366 {
4367   \bool_if:NTF \l_@@_initial_open_bool
4368     { \@@_open_x_initial_dim: }
4369     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4370   \bool_if:NTF \l_@@_final_open_bool
4371     { \@@_open_x_final_dim: }
4372     { \@@_set_final_coords_from_anchor:n { mid-west } }
4373   \bool_lazy_and:nnTF
4374     \l_@@_initial_open_bool
4375     \l_@@_final_open_bool
4376   {
4377     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4378     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4379     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4380     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4381     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4382   }
4383   {
4384     \bool_if:NT \l_@@_initial_open_bool
4385       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4386     \bool_if:NT \l_@@_final_open_bool
4387       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4388   }
4389   \@@_draw_line:
4390 }

4391 \cs_new_protected:Npn \@@_open_y_initial_dim:
4392 {
4393   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4394   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4395   {
4396     \cs_if_exist:cT
4397       { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4398     {
4399       \pgfpointanchor
4400         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4401         { north }
4402       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4403       { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4404     }
4405   }
4406   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4407   {
4408     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4409     \dim_set:Nn \l_@@_y_initial_dim
4410     {
4411       \fp_to_dim:n
4412       {
4413         \pgf@y
4414         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4415       }
4416     }
4417   }
4418 }

```

```

4419 \cs_new_protected:Npn \@@_open_y_final_dim:
4420 {
4421   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4422   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4423   {
4424     \cs_if_exist:cT
4425     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4426     {
4427       \pgfpointanchor
4428       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4429       { south }
4430       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4431       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4432     }
4433   }
4434   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4435   {
4436     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4437     \dim_set:Nn \l_@@_y_final_dim
4438     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4439   }
4440 }

```

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.

```

4441 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4442 {
4443   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4444   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4445   {
4446     \@@_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.

```

4447   \group_begin:
4448   \@@_open_shorten:
4449   \int_if_zero:nTF { #2 }
4450   { \color { nicematrix-first-col } }
4451   {
4452     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4453     { \color { nicematrix-last-col } }
4454   }
4455   \keys_set:nn { nicematrix / xdots } { #3 }
4456   \@@_color:o \l_@@_xdots_color_tl
4457   \@@_actually_draw_Vdots:
4458   \group_end:
4459 }
4460 }

```

The command `\@@_actually_draw_Vdots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool`.

The following function is also used by `\Vdotsfor`.

```

4461 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4462 {

```

First, the case of a dotted line open on both sides.

```
4463 \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.

```
4464 {
4465   \@@_open_y_initial_dim:
4466   \@@_open_y_final_dim:
4467   \int_if_zero:nTF \l_@@_initial_j_int
```

We have a dotted line open on both sides in the “first column”.

```
4468 {
4469   \@@_qpoint:n { col - 1 }
4470   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4471   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4472   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4473   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4474 }
4475 {
4476   \bool_lazy_and:nnTF
4477   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4478   { \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”.

```
4479 {
4480   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4481   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4482   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4483   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4484   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4485 }
```

We have a dotted line open on both sides which is *not* in an exterior column.

```
4486 {
4487   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4488   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4489   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4490   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4491 }
4492 }
4493 }
```

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.

```
4494 {
4495   \bool_set_false:N \l_tmpa_bool
4496   \bool_if:NF \l_@@_initial_open_bool
4497   {
4498     \bool_if:NF \l_@@_final_open_bool
4499     {
4500       \@@_set_initial_coords_from_anchor:n { south-west }
4501       \@@_set_final_coords_from_anchor:n { north-west }
4502       \bool_set:Nn \l_tmpa_bool
4503       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4504     }
4505   }
```

Now, we try to determine whether the column is of type c or may be considered as if.

```
4506 \bool_if:NTF \l_@@_initial_open_bool
4507 {
4508   \@@_open_y_initial_dim:
4509   \@@_set_final_coords_from_anchor:n { north }
4510   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4511 }
4512 {
4513   \@@_set_initial_coords_from_anchor:n { south }
4514   \bool_if:NTF \l_@@_final_open_bool
```

```
4515 \@@_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.

```
4516 {
4517   \@@_set_final_coords_from_anchor:n { north }
4518   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4519   {
4520     \dim_set:Nn \l_@@_x_initial_dim
4521     {
4522       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4523       \l_@@_x_initial_dim \l_@@_x_final_dim
4524     }
4525   }
4526 }
4527 }
4528 }
4529 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4530 \@@_draw_line:
4531 }
```

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.

```
4532 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4533 {
4534   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4535   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4536   {
4537     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 1
```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```
4538   \group_begin:
4539   \@@_open_shorten:
4540   \keys_set:nn { nicematrix / xdots } { #3 }
4541   \@@_color:o \l_@@_xdots_color_tl
4542   \@@_actually_draw_Ddots:
4543   \group_end:
4544 }
4545 }
```

The command `\@@_actually_draw_Ddots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

```
4546 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4547 {
4548   \bool_if:NTF \l_@@_initial_open_bool
4549   {
4550     \@@_open_y_initial_dim:
4551     \@@_open_x_initial_dim:
```

```

4552     }
4553     { \@@_set_initial_coords_from_anchor:n { south-east } }
4554     \bool_if:NTF \l_@@_final_open_bool
4555     {
4556         \@@_open_x_final_dim:
4557         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4558     }
4559     { \@@_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.

```

4560     \bool_if:NT \l_@@_parallelize_diags_bool
4561     {
4562         \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).

```

4563         \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.

```

4564         {
4565             \dim_gset:Nn \g_@@_delta_x_one_dim
4566             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4567             \dim_gset:Nn \g_@@_delta_y_one_dim
4568             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4569         }

```

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`.

```

4570         {
4571             \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4572             {
4573                 \dim_set:Nn \l_@@_y_final_dim
4574                 {
4575                     \l_@@_y_initial_dim +
4576                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4577                     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4578                 }
4579             }
4580         }
4581     }
4582     \@@_draw_line:
4583 }

```

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.

```

4584 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4585 {
4586     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4587     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4588     {
4589         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 1 { -1 }

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

4590     \group_begin:
4591     \@@_open_shorten:
4592     \keys_set:nn { nicematrix / xdots } { #3 }
4593     \@@_color:o \l_@@_xdots_color_tl
4594     \@@_actually_draw_Iddots:
4595     \group_end:
4596 }
4597 }

```


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.`

```

4598 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4599 {
4600   \bool_if:NTF \l_@@_initial_open_bool
4601   {
4602     \@@_open_y_initial_dim:
4603     \@@_open_x_initial_dim:
4604   }
4605   { \@@_set_initial_coords_from_anchor:n { south-west } }
4606   \bool_if:NTF \l_@@_final_open_bool
4607   {
4608     \@@_open_y_final_dim:
4609     \@@_open_x_final_dim:
4610   }
4611   { \@@_set_final_coords_from_anchor:n { north-east } }
4612   \bool_if:NT \l_@@_parallelize_diags_bool
4613   {
4614     \int_gincr:N \g_@@_iddots_int
4615     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4616     {
4617       \dim_gset:Nn \g_@@_delta_x_two_dim
4618       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4619       \dim_gset:Nn \g_@@_delta_y_two_dim
4620       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4621     }
4622     {
4623       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4624       {
4625         \dim_set:Nn \l_@@_y_final_dim
4626         {
4627           \l_@@_y_initial_dim +
4628           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4629           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4630         }
4631       }
4632     }
4633   }
4634   \@@_draw_line:
4635 }

```

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`

```

4636 \cs_new_protected:Npn \@@_draw_line:
4637 {
4638   \pgfrememberpicturerepositiononpagetrue
4639   \pgf@relevantforpicturesizefalse
4640   \bool_lazy_or:nnTF
4641     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4642     \l_@@_dotted_bool
4643     \@@_draw_standard_dotted_line:
4644     \@@_draw_unstandard_dotted_line:
4645 }

```

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.

```

4646 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4647 {
4648   \begin { scope }
4649     \@@_draw_unstandard_dotted_line:o
4650     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4651 }

```

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.

```

4652 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4653 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4654 {
4655   \@@_draw_unstandard_dotted_line:nooo
4656   { #1 }
4657   \l_@@_xdots_up_tl
4658   \l_@@_xdots_down_tl
4659   \l_@@_xdots_middle_tl
4660 }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4661 \hook_gput_code:nnn { begindocument } { . }
4662 {
4663   \IfPackageLoadedT { tikz }
4664   {
4665     \tikzset
4666     {
4667       @@_node_above / .style = { sloped , above } ,
4668       @@_node_below / .style = { sloped , below } ,
4669       @@_node_middle / .style =
4670       {
4671         sloped ,
4672         inner~sep = \c_@@_innersep_middle_dim
4673       }
4674     }
4675   }
4676 }

```

```

4677 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4678 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4679 {

```

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.

```

4680 \dim_zero_new:N \l_@@_l_dim
4681 \dim_set:Nn \l_@@_l_dim
4682 {
4683   \fp_to_dim:n
4684   {
4685     sqrt
4686     (
4687       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4688       +
4689       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4690     )
4691   }
4692 }

```

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.

```

4693 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4694 {
4695   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4696   \@@_draw_unstandard_dotted_line_i:
4697 }

```

If the key `xdots/horizontal-labels` has been used.

```

4698 \bool_if:NT \l_@@_xdots_h_labels_bool
4699 {
4700   \tikzset
4701   {
4702     @@_node_above / .style = { auto = left } ,
4703     @@_node_below / .style = { auto = right } ,
4704     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4705   }
4706 }
4707 \tl_if_empty:nF { #4 }
4708 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4709 \draw
4710 [ #1 ]
4711 ( \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).

```

4712 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4713 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4714 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4715 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4716 \end { scope }
4717 }
4718 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4719 {
4720   \dim_set:Nn \l_tmpa_dim
4721   {
4722     \l_@@_x_initial_dim
4723     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4724     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4725   }
4726   \dim_set:Nn \l_tmpb_dim
4727   {
4728     \l_@@_y_initial_dim
4729     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4730     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4731   }
4732   \dim_set:Nn \l_@@_tmpc_dim
4733   {
4734     \l_@@_x_final_dim
4735     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4736     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4737   }
4738   \dim_set:Nn \l_@@_tmpd_dim
4739   {
4740     \l_@@_y_final_dim
4741     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4742     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4743   }
4744   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4745   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4746   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4747   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4748 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4749 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4750 {
4751   \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.

```

4752   \dim_zero_new:N \l_@@_l_dim
4753   \dim_set:Nn \l_@@_l_dim
4754   {
4755     \fp_to_dim:n
4756     {
4757       sqrt
4758       (
4759         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4760         +
4761         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4762       )
4763     }
4764   }

```

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.

```

4765   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4766   {
4767     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4768     \@@_draw_standard_dotted_line_i:
4769   }
4770   \group_end:
4771   \bool_lazy_all:nF
4772   {
4773     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4774     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4775     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4776     }
4777     \l_@@_labels_standard_dotted_line:
4778 }
4779 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4780 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4781 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

4782     \int_set:Nn \l_1tmpa_int
4783     {
4784         \dim_ratio:nn
4785         {
4786             \l_@@_l_dim
4787             - \l_@@_xdots_shorten_start_dim
4788             - \l_@@_xdots_shorten_end_dim
4789         }
4790         \l_@@_xdots_inter_dim
4791     }

```

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.

```

4792     \dim_set:Nn \l_1tmpa_dim
4793     {
4794         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4795         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4796     }
4797     \dim_set:Nn \l_1tmpb_dim
4798     {
4799         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4800         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4801     }

```

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.

```

4802     \dim_gadd:Nn \l_@@_x_initial_dim
4803     {
4804         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4805         \dim_ratio:nn
4806         {
4807             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4808             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4809         }
4810         { 2 \l_@@_l_dim }
4811     }
4812     \dim_gadd:Nn \l_@@_y_initial_dim
4813     {
4814         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4815         \dim_ratio:nn
4816         {
4817             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4818             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4819         }
4820         { 2 \l_@@_l_dim }
4821     }
4822     \pgf@relevantforpicturesizefalse
4823     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4824     {
4825         \pgfpathcircle
4826         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4827         { \l_@@_xdots_radius_dim }
4828         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4829         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4830     }

```

```

4831 \pgfusepathqfill
4832 }

4833 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4834 {
4835   \pgfscope
4836   \pgftransformshift
4837   {
4838     \pgfpointlineattime { 0.5 }
4839     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4840     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4841   }
4842   \fp_set:Nn \l_tmpa_fp
4843   {
4844     atand
4845     (
4846       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4847       \l_@@_x_final_dim - \l_@@_x_initial_dim
4848     )
4849   }
4850   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4851   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4852   \tl_if_empty:NF \l_@@_xdots_middle_tl
4853   {
4854     \begin { pgfscope }
4855     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4856     \pgfnode
4857     { rectangle }
4858     { center }
4859     {
4860       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4861       {
4862         \c_math_toggle_token
4863         \scriptstyle \l_@@_xdots_middle_tl
4864         \c_math_toggle_token
4865       }
4866     }
4867     { }
4868     {
4869       \pgfsetfillcolor { white }
4870       \pgfusepath { fill }
4871     }
4872     \end { pgfscope }
4873   }
4874   \tl_if_empty:NF \l_@@_xdots_up_tl
4875   {
4876     \pgfnode
4877     { rectangle }
4878     { south }
4879     {
4880       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4881       {
4882         \c_math_toggle_token
4883         \scriptstyle \l_@@_xdots_up_tl
4884         \c_math_toggle_token
4885       }
4886     }
4887     { }
4888     { \pgfusepath { } }
4889   }
4890   \tl_if_empty:NF \l_@@_xdots_down_tl
4891   {
4892     \pgfnode

```

```

4893     { rectangle }
4894     { north }
4895     {
4896       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4897       {
4898         \c_math_toggle_token
4899         \scriptstyle \l_@@_xdots_down_tl
4900         \c_math_toggle_token
4901       }
4902     }
4903     { }
4904     { \pgfusepath { } }
4905   }
4906 \endpgfscope
4907 }

```

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.

```

4908 \hook_gput_code:nnn { begindocument } { . }
4909 {
4910   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4911   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4912   \cs_new_protected:Npn \@@_Ldots
4913     { \@@_collect_options:n { \@@_Ldots_i } }
4914   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4915     {
4916       \int_if_zero:nTF \c@jCol
4917       { \@@_error:nn { in~first~col } \Ldots }
4918       {
4919         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4920         { \@@_error:nn { in~last~col } \Ldots }
4921         {
4922           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4923           { #1 , down = #2 , up = #3 , middle = #4 }
4924         }
4925       }
4926       \bool_if:NF \l_@@_nullify_dots_bool
4927       { \phantom { \ensuremath { \@@_old_ldots } } }
4928       \bool_gset_true:N \g_@@_empty_cell_bool
4929     }

4930   \cs_new_protected:Npn \@@_Cdots
4931     { \@@_collect_options:n { \@@_Cdots_i } }
4932   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4933     {
4934       \int_if_zero:nTF \c@jCol
4935       { \@@_error:nn { in~first~col } \Cdots }
4936       {
4937         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```

```

4938         { \@@_error:nn { in~last~col } \Cdots }
4939     {
4940         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4941         { #1 , down = #2 , up = #3 , middle = #4 }
4942     }
4943 }
4944 \bool_if:NF \l_@@_nullify_dots_bool
4945 { \phantom { \ensuremath { \@@_old_cdots } } }
4946 \bool_gset_true:N \g_@@_empty_cell_bool
4947 }

4948 \cs_new_protected:Npn \@@_Vdots
4949 { \@@_collect_options:n { \@@_Vdots_i } }
4950 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4951 {
4952     \int_if_zero:nTF \c@iRow
4953     { \@@_error:nn { in~first~row } \Vdots }
4954     {
4955         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4956         { \@@_error:nn { in~last~row } \Vdots }
4957         {
4958             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4959             { #1 , down = #2 , up = #3 , middle = #4 }
4960         }
4961     }
4962     \bool_if:NF \l_@@_nullify_dots_bool
4963     { \phantom { \ensuremath { \@@_old_vdots } } }
4964     \bool_gset_true:N \g_@@_empty_cell_bool
4965 }

4966 \cs_new_protected:Npn \@@_Ddots
4967 { \@@_collect_options:n { \@@_Ddots_i } }
4968 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4969 {
4970     \int_case:nnF \c@iRow
4971     {
4972         0 { \@@_error:nn { in~first~row } \Ddots }
4973         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4974     }
4975     {
4976         \int_case:nnF \c@jCol
4977         {
4978             0 { \@@_error:nn { in~first~col } \Ddots }
4979             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4980         }
4981         {
4982             \keys_set_known:nn { nicematrix / Ddots } { #1 }
4983             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4984             { #1 , down = #2 , up = #3 , middle = #4 }
4985         }
4986     }
4987 }
4988 \bool_if:NF \l_@@_nullify_dots_bool
4989 { \phantom { \ensuremath { \@@_old_ddots } } }
4990 \bool_gset_true:N \g_@@_empty_cell_bool
4991 }

4992 \cs_new_protected:Npn \@@_Iddots
4993 { \@@_collect_options:n { \@@_Iddots_i } }
4994 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4995 {

```



```

4996 \int_case:nnF \c@iRow
4997 {
4998     0 { \@@_error:nn { in~first~row } \Iddots }
4999     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5000 }
5001 {
5002     \int_case:nnF \c@jCol
5003     {
5004         0 { \@@_error:nn { in~first~col } \Iddots }
5005         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5006     }
5007     {
5008         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5009         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5010         { #1 , down = #2 , up = #3 , middle = #4 }
5011     }
5012 }
5013 \bool_if:NF \l_@@_nullify_dots_bool
5014 { \phantom { \ensuremath { \@@_old_iddots } } }
5015 \bool_gset_true:N \g_@@_empty_cell_bool
5016 }
5017 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5018 \keys_define:nn { nicematrix / Ddots }
5019 {
5020     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5021     draw-first .default:n = true ,
5022     draw-first .value_forbidden:n = true
5023 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5024 \cs_new_protected:Npn \@@_Hspace:
5025 {
5026     \bool_gset_true:N \g_@@_empty_cell_bool
5027     \hspace
5028 }

```

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`.

```

5029 \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`.

```

5030 \cs_new:Npn \@@_Hdotsfor:
5031 {
5032     \bool_lazy_and:nnTF
5033     { \int_if_zero_p:n \c@jCol }
5034     { \int_if_zero_p:n \l_@@_first_col_int }
5035     {
5036         \bool_if:NTF \g_@@_after_col_zero_bool
5037         {
5038             \multicolumn { 1 } { c } { }
5039             \@@_Hdotsfor_i
5040         }
5041         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5042     }
5043 }

```

```

5044     \multicolumn { 1 } { c } { }
5045     \@@_Hdotsfor_i
5046   }
5047 }

```

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:`).

```

5048 \hook_gput_code:nnn { begindocument } { . }
5049 {
5050   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5051   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put `!` before the last optionnal argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```

5052   \cs_new_protected:Npn \@@_Hdotsfor_i
5053     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5054   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5055     {
5056       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5057       {
5058         \@@_Hdotsfor:nnnn
5059         { \int_use:N \c@iRow }
5060         { \int_use:N \c@jCol }
5061         { #2 }
5062         {
5063           #1 , #3 ,
5064           down = \exp_not:n { #4 } ,
5065           up = \exp_not:n { #5 } ,
5066           middle = \exp_not:n { #6 }
5067         }
5068       }
5069       \prg_replicate:nn { #2 - 1 }
5070       {
5071         &
5072         \multicolumn { 1 } { c } { }
5073         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5074       }
5075     }
5076 }

```

```

5077 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5078 {
5079   \bool_set_false:N \l_@@_initial_open_bool
5080   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5081   \int_set:Nn \l_@@_initial_i_int { #1 }
5082   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5083   \int_compare:nNnTF { #2 } = \c_one_int
5084   {
5085     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5086     \bool_set_true:N \l_@@_initial_open_bool
5087   }
5088   {
5089     \cs_if_exist:cTF
5090     {
5091       pgf @ sh @ ns @ \@@_env:
5092       - \int_use:N \l_@@_initial_i_int
5093       - \int_eval:n { #2 - 1 }
5094     }
5095     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5096     {

```

```

5097         \int_set:Nn \l_@@_initial_j_int { #2 }
5098         \bool_set_true:N \l_@@_initial_open_bool
5099     }
5100 }
5101 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5102 {
5103     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5104     \bool_set_true:N \l_@@_final_open_bool
5105 }
5106 {
5107     \cs_if_exist:cTF
5108     {
5109         pgf @ sh @ ns @ \@@_env:
5110         - \int_use:N \l_@@_final_i_int
5111         - \int_eval:n { #2 + #3 }
5112     }
5113     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5114     {
5115         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5116         \bool_set_true:N \l_@@_final_open_bool
5117     }
5118 }
5119 \group_begin:
5120 \@@_open_shorten:
5121 \int_if_zero:nTF { #1 }
5122 { \color { nicematrix-first-row } }
5123 {
5124     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5125     { \color { nicematrix-last-row } }
5126 }
5127
5128 \keys_set:nn { nicematrix / xdots } { #4 }
5129 \@@_color:o \l_@@_xdots_color_tl
5130 \@@_actually_draw_Ldots:
5131 \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).

```

5132     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5133     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5134 }

5135 \hook_gput_code:nnn { begindocument } { . }
5136 {
5137     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5138     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5139     \cs_new_protected:Npn \@@_Vdotsfor:
5140     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5141     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5142     {
5143         \bool_gset_true:N \g_@@_empty_cell_bool
5144         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5145         {
5146             \@@_Vdotsfor:nnnn
5147             { \int_use:N \c@iRow }
5148             { \int_use:N \c@jCol }
5149             { #2 }
5150             {
5151                 #1 , #3 ,
5152                 down = \exp_not:n { #4 } ,
5153                 up = \exp_not:n { #5 } ,

```

```

5154         middle = \exp_not:n { #6 }
5155     }
5156 }
5157 }
5158 }

```

```

5159 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5160 {
5161     \bool_set_false:N \l_@@_initial_open_bool
5162     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5163     \int_set:Nn \l_@@_initial_j_int { #2 }
5164     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5165     \int_compare:nNnTF { #1 } = \c_one_int
5166     {
5167         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5168         \bool_set_true:N \l_@@_initial_open_bool
5169     }
5170     {
5171         \cs_if_exist:cTF
5172         {
5173             pgf @ sh @ ns @ \@@_env:
5174             - \int_eval:n { #1 - 1 }
5175             - \int_use:N \l_@@_initial_j_int
5176         }
5177         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5178         {
5179             \int_set:Nn \l_@@_initial_i_int { #1 }
5180             \bool_set_true:N \l_@@_initial_open_bool
5181         }
5182     }
5183     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5184     {
5185         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5186         \bool_set_true:N \l_@@_final_open_bool
5187     }
5188     {
5189         \cs_if_exist:cTF
5190         {
5191             pgf @ sh @ ns @ \@@_env:
5192             - \int_eval:n { #1 + #3 }
5193             - \int_use:N \l_@@_final_j_int
5194         }
5195         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5196         {
5197             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5198             \bool_set_true:N \l_@@_final_open_bool
5199         }
5200     }
5201     \group_begin:
5202     \@@_open_shorten:
5203     \int_if_zero:nTF { #2 }
5204     { \color { nicematrix-first-col } }
5205     {
5206         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5207         { \color { nicematrix-last-col } }
5208     }
5209     \keys_set:nn { nicematrix / xdots } { #4 }
5210     \@@_color:o \l_@@_xdots_color_tl
5211     \@@_actually_draw_Vdots:
5212     \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).

```

5213 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5214 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5215 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5216 \NewDocumentCommand \@@_rotate: { 0 { } }
5217 {
5218   \peek_remove_spaces:n
5219   {
5220     \bool_gset_true:N \g_@@_rotate_bool
5221     \keys_set:nn { nicematrix / rotate } { #1 }
5222   }
5223 }

5224 \keys_define:nn { nicematrix / rotate }
5225 {
5226   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5227   c .value_forbidden:n = true ,
5228   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5229 }

```

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

```

5230 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5231 {
5232   \tl_if_empty:nTF { #2 }
5233   { #1 }
5234   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5235 }
5236 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5237 { \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).

```

5238 \hook_gput_code:nnn { begindocument } { . }
5239 {

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5240 \cs_set_nopar:Npn \l_@@_argspec_tl
5241 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5242 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5243 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5244 {
5245   \group_begin:
5246   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5247   \@@_color:o \l_@@_xdots_color_tl
5248   \use:e
5249   {
5250     \@@_line_i:nn
5251     { \@@_double_int_eval:n #2 - \q_stop }
5252     { \@@_double_int_eval:n #3 - \q_stop }
5253   }
5254   \group_end:
5255 }
5256 }
5257 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5258 {
5259   \bool_set_false:N \l_@@_initial_open_bool
5260   \bool_set_false:N \l_@@_final_open_bool
5261   \bool_lazy_or:nnTF
5262   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5263   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5264   { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5265 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5266 }
5267 \hook_gput_code:nnn { begindocument } { . }
5268 {
5269   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5270   {

```

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:`.

```

5271 \c_@@_pgfortikzpicture_tl
5272 \@@_draw_line_iii:nn { #1 } { #2 }
5273 \c_@@_endpgfortikzpicture_tl
5274 }
5275 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5276 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5277 {
5278   \pgfrememberpicturepositiononpagetrue
5279   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5280   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5281   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5282   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5283   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5284   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5285   \@@_draw_line:
5286 }

```

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

```
5287 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5288 { \int_compare:nNtT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times in `\RowStyle`.

```
5289 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5290 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5291 {
5292   \tl_gput_right:Ne \g_@@_row_style_tl
5293   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5294   \exp_not:N
5295   \@@_if_row_less_than:nn
5296   { \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`).

```
5297   { \exp_not:n { #1 } \scan_stop: }
5298 }
5299 }
```

```
5300 \keys_define:nn { nicematrix / RowStyle }
5301 {
5302   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5303   cell-space-top-limit .value_required:n = true ,
5304   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5305   cell-space-bottom-limit .value_required:n = true ,
5306   cell-space-limits .meta:n =
5307   {
5308     cell-space-top-limit = #1 ,
5309     cell-space-bottom-limit = #1 ,
5310   } ,
5311   color .tl_set:N = \l_@@_color_tl ,
5312   color .value_required:n = true ,
5313   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5314   bold .default:n = true ,
5315   nb-rows .code:n =
5316   { \str_if_eq:eeTF { #1 } { * }
5317     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5318     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5319   nb-rows .value_required:n = true ,
5320   fill .tl_set:N = \l_@@_fill_tl ,
5321   fill .value_required:n = true ,
5322   opacity .tl_set:N = \l_@@_opacity_tl ,
5323   opacity .value_required:n = true ,
5324   rowcolor .tl_set:N = \l_@@_fill_tl ,
5325   rowcolor .value_required:n = true ,
5326   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
```

```

5327     rounded-corners .default:n = 4 pt ,
5328     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5329 }

```

```

5330 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5331 {
5332   \group_begin:
5333   \tl_clear:N \l_@@_fill_tl
5334   \tl_clear:N \l_@@_opacity_tl
5335   \tl_clear:N \l_@@_color_tl
5336   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5337   \dim_zero:N \l_@@_rounded_corners_dim
5338   \dim_zero:N \l_tmpa_dim
5339   \dim_zero:N \l_tmpb_dim
5340   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` (of its alias `fill`) has been used.

```

5341   \tl_if_empty:NF \l_@@_fill_tl
5342   {
5343     \@@_add_opacity_to_fill:
5344     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5345     {

```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```

5346       \int_compare:nNnTF \c_jCol > \c_one_int
5347       {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row). The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5348         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5349         { \int_use:N \c_iRow - \int_use:N \c_jCol }
5350         { \int_use:N \c_iRow - * }
5351         { \dim_use:N \l_@@_rounded_corners_dim }

```

Then, the other rows (if there are several rows).

```

5352         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5353         { \@@_rounded_from_row:n { \c_iRow + 1 } }
5354     }

```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```

5355     { \@@_rounded_from_row:n { \c_iRow } }
5356   }
5357 }
5358 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5359   \dim_compare:nNnT \l_tmpa_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       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5365         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5366         { \dim_use:N \l_tmpa_dim }
5367     }
5368   }
5369 }

```


`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5370   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5371   {
5372     \@@_put_in_row_style:e
5373     {
5374       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5375       {
5376         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5377         { \dim_use:N \l_tmpb_dim }
5378       }
5379     }
5380   }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5381   \tl_if_empty:NF \l_@@_color_tl
5382   {
5383     \@@_put_in_row_style:e
5384     {
5385       \mode_leave_vertical:
5386       \@@_color:n { \l_@@_color_tl }
5387     }
5388   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5389   \bool_if:NT \l_@@_bold_row_style_bool
5390   {
5391     \@@_put_in_row_style:n
5392     {
5393       \exp_not:n
5394       {
5395         \if_mode_math:
5396         \c_math_toggle_token
5397         \bfseries \boldmath
5398         \c_math_toggle_token
5399         \else:
5400         \bfseries \boldmath
5401         \fi:
5402       }
5403     }
5404   }
5405   \group_end:
5406   \g_@@_row_style_tl
5407   \ignorespaces
5408 }

```

The following commande must *not* be protected.

```

5409 \cs_new:Npn \@@_rounded_from_row:n #1
5410 {
5411   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5412   { \int_eval:n { #1 } - 1 }
5413   {
5414     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5415     - \exp_not:n { \int_use:N \c@jCol }
5416   }
5417   { \dim_use:N \l_@@_rounded_corners_dim }
5418 }

```

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).

```
5419 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5420 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5421 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5422 {
```

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.

```
5423 \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`.

```
5424 \str_if_in:nnF { #1 } { !! }
5425 {
5426 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5427 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5428 }
5429 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5430 {
5431 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5432 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5433 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5434 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5435 }
```

The following command must be used within a `\pgfpicture`.

```
5436 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5437 {
5438 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5439 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5440     \group_begin:
5441     \pgfsetcornersarced
5442     {
5443         \pgfpoint
5444         { \l_@@_tab_rounded_corners_dim }
5445         { \l_@@_tab_rounded_corners_dim }
5446     }

```

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.

```

5447     \bool_if:NTF \l_@@_hvlines_bool
5448     {
5449         \pgfpathrectanglecorners
5450         {
5451             \pgfpointadd
5452             { \@@_qpoint:n { row-1 } }
5453             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5454         }
5455         {
5456             \pgfpointadd
5457             {
5458                 \@@_qpoint:n
5459                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5460             }
5461             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5462         }
5463     }
5464     {
5465         \pgfpathrectanglecorners
5466         { \@@_qpoint:n { row-1 } }
5467         {
5468             \pgfpointadd
5469             {
5470                 \@@_qpoint:n
5471                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5472             }
5473             { \pgfpoint \c_zero_dim \arrayrulewidth }
5474         }
5475     }
5476     \pgfusepath { clip }
5477     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5478     }
5479 }

```

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`).

```

5480 \cs_new_protected:Npn \@@_actually_color:
5481 {
5482     \pgfpicture
5483     \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.

```

5484     \@@_clip_with_rounded_corners:
5485     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5486     {
5487         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5488     {
5489         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5490         \use:c { g_@@_color _ 1 _tl }
5491         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5492     }
5493     {
5494         \begin { pgfscope }
5495             \@@_color_opacity ##2
5496             \use:c { g_@@_color _ ##1 _tl }
5497             \tl_gclear:c { g_@@_color _ ##1 _tl }
5498             \pgfusepath { fill }
5499         \end { pgfscope }
5500     }
5501 }
5502 \endpgfpicture
5503 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5504 \cs_new_protected:Npn \@@_color_opacity
5505 {
5506     \peek_meaning:NTF [
5507         { \@@_color_opacity:w }
5508         { \@@_color_opacity:w [ ] }
5509     }

```

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 currying.

```

5510 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5511 {
5512     \tl_clear:N \l_tmpa_tl
5513     \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.

```

5514     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5515     \tl_if_empty:NTF \l_tmpb_tl
5516         { \@declaredcolor }
5517         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5518 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5519 \keys_define:nn { nicematrix / color-opacity }
5520 {
5521     opacity .tl_set:N          = \l_tmpa_tl ,
5522     opacity .value_required:n = true
5523 }

5524 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5525 {
5526     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5527     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5528     \@@_cartesian_path:
5529 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5530 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5531 {
5532     \tl_if_blank:nF { #2 }
5533     {

```

```

5534 \@@_add_to_colors_seq:en
5535 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5536 { \@@_cartesian_color:nn { #3 } { - } }
5537 }
5538 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5539 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5540 {
5541   \tl_if_blank:nF { #2 }
5542   {
5543     \@@_add_to_colors_seq:en
5544     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5545     { \@@_cartesian_color:nn { - } { #3 } }
5546   }
5547 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5548 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5549 {
5550   \tl_if_blank:nF { #2 }
5551   {
5552     \@@_add_to_colors_seq:en
5553     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5554     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5555   }
5556 }

```

The last argument is the radius of the corners of the rectangle.

```

5557 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5558 {
5559   \tl_if_blank:nF { #2 }
5560   {
5561     \@@_add_to_colors_seq:en
5562     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5563     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5564   }
5565 }

```

The last argument is the radius of the corners of the rectangle.

```

5566 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5567 {
5568   \@@_cut_on_hyphen:w #1 \q_stop
5569   \tl_clear_new:N \l_@@_tmpc_tl
5570   \tl_clear_new:N \l_@@_tmpd_tl
5571   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5572   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5573   \@@_cut_on_hyphen:w #2 \q_stop
5574   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5575   \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`.

```

5576 \@@_cartesian_path:n { #3 }
5577 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5578 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5579 {
5580   \clist_map_inline:nn { #3 }
5581   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5582 }

```

```

5583 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5584 {
5585   \int_step_inline:nn \c@iRow
5586   {
5587     \int_step_inline:nn \c@jCol
5588     {
5589       \int_if_even:nTF { #####1 + ##1 }
5590       { \@@_cellcolor [ #1 ] { #2 } }
5591       { \@@_cellcolor [ #1 ] { #3 } }
5592       { ##1 - #####1 }
5593     }
5594   }
5595 }

```

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”.

```

5596 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5597 {
5598   \@@_rectanglecolor [ #1 ] { #2 }
5599   { 1 - 1 }
5600   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5601 }

```

```

5602 \keys_define:nn { nicematrix / rowcolors }
5603 {
5604   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5605   respect-blocks .default:n = true ,
5606   cols .tl_set:N = \l_@@_cols_tl ,
5607   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5608   restart .default:n = true ,
5609   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5610 }

```

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*.

```

5611 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5612 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5613   \group_begin:
5614   \seq_clear_new:N \l_@@_colors_seq
5615   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5616   \tl_clear_new:N \l_@@_cols_tl
5617   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5618   \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).

```

5619   \int_zero_new:N \l_@@_color_int
5620   \int_set_eq:NN \l_@@_color_int \c_one_int
5621   \bool_if:NT \l_@@_respect_blocks_bool
5622   {

```

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`).

```

5623     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5624     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5625     { \@@_not_in_exterior_p:nnnnn ##1 }
5626   }
5627   \pgfpicture
5628   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5629   \clist_map_inline:nn { #2 }
5630   {
5631     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5632     \tl_if_in:NnTF \l_tmpa_tl { - }
5633     { \@@_cut_on_hyphen:w ##1 \q_stop }
5634     { \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.

```

5635     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5636     \int_set:Nn \l_@@_color_int
5637     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5638     \int_zero_new:N \l_@@_tmpc_int
5639     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5640     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5641     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5642         \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).

```

5643         \bool_if:NT \l_@@_respect_blocks_bool
5644         {
5645           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5646           { \@@_intersect_our_row_p:nnnnn #####1 }
5647           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5648         }
5649         \tl_set:No \l_@@_rows_tl
5650         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5651         \tl_clear_new:N \l_@@_color_tl
5652         \tl_set:Ne \l_@@_color_tl
5653         {
5654           \@@_color_index:n
5655           {
5656             \int_mod:nn
5657             { \l_@@_color_int - 1 }
5658             { \seq_count:N \l_@@_colors_seq }
5659             + 1
5660           }
5661         }
5662         \tl_if_empty:NF \l_@@_color_tl
5663         {
5664           \@@_add_to_colors_seq:ee
5665           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5666           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5667         }
5668         \int_incr:N \l_@@_color_int
5669         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5670       }
5671     }
5672   \endpgfpicture

```

```

5673 \group_end:
5674 }

```

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.

```

5675 \cs_new:Npn \@@_color_index:n #1
5676 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5677 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5678 { \@@_color_index:n { #1 - 1 } }
5679 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5680 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5681 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5682 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5683 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5684 {
5685 \int_compare:nNnT { #3 } > \l_tmpb_int
5686 { \int_set:Nn \l_tmpb_int { #3 } }
5687 }

```

```

5688 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5689 {
5690 \int_if_zero:nTF { #4 }
5691 \prg_return_false:
5692 {
5693 \int_compare:nNnTF { #2 } > \c_jCol
5694 \prg_return_false:
5695 \prg_return_true:
5696 }
5697 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5698 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5699 {
5700 \int_compare:nNnTF { #1 } > \l_tmpa_int
5701 \prg_return_false:
5702 {
5703 \int_compare:nNnTF \l_tmpa_int > { #3 }
5704 \prg_return_false:
5705 \prg_return_true:
5706 }
5707 }

```

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`).

```

5708 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5709 {
5710 \dim_compare:nNnTF { #1 } = \c_zero_dim

```



```

5711 {
5712   \bool_if:NTF
5713     \l_@@_nocolor_used_bool
5714     \@@_cartesian_path_normal_ii:
5715   {
5716     \clist_if_empty:NTF \l_@@_corners_cells_clist
5717       { \@@_cartesian_path_normal_i:n { #1 } }
5718       \@@_cartesian_path_normal_ii:
5719   }
5720 }
5721 { \@@_cartesian_path_normal_i:n { #1 } }
5722 }

```

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.

```

5723 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5724 {
5725   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5726   \clist_map_inline:Nn \l_@@_cols_tl
5727   {
5728     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5729     \tl_if_in:NnTF \l_tmpa_tl { - }
5730       { \@@_cut_on_hyphen:w ##1 \q_stop }
5731       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5732     \tl_if_empty:NTF \l_tmpa_tl
5733       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5734       {
5735         \str_if_eq:eeT \l_tmpa_tl { * }
5736         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5737       }
5738     \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5739       { \@@_error:n { Invalid~col~number } }
5740     \tl_if_empty:NTF \l_tmpb_tl
5741       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5742       {
5743         \str_if_eq:eeT \l_tmpb_tl { * }
5744         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5745       }
5746     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5747       { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5748     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5749     \@@_qpoint:n { col - \l_tmpa_tl }
5750     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5751       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5752       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5753     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5754     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5755   \clist_map_inline:Nn \l_@@_rows_tl
5756   {
5757     \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5758     \tl_if_in:NnTF \l_tmpa_tl { - }
5759       { \@@_cut_on_hyphen:w #####1 \q_stop }
5760       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5761     \tl_if_empty:NTF \l_tmpa_tl
5762       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5763       {
5764         \str_if_eq:eeT \l_tmpa_tl { * }
5765         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }

```

```

5766     }
5767     \tl_if_empty:NTF \l_tmpb_tl
5768     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5769     {
5770         \str_if_eq:eeT \l_tmpb_tl { * }
5771         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5772     }
5773     \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5774     { \@@_error:n { Invalid~row~number } }
5775     \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5776     { \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.

```

5777     \cs_if_exist:cF
5778     { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5779     {
5780         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5781         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5782         \@@_qpoint:n { row - \l_tmpa_tl }
5783         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5784         \pgfpathrectanglecorners
5785         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5786         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5787     }
5788 }
5789 }
5790 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key **corners** is used).

```

5791 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5792 {
5793     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5794     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5795     \clist_map_inline:Nn \l_@@_cols_tl
5796     {
5797         \@@_qpoint:n { col - ##1 }
5798         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5799         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5800         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5801         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5802         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5803     \clist_map_inline:Nn \l_@@_rows_tl
5804     {
5805         \@@_if_in_corner:nF { #####1 - ##1 }
5806         {
5807             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5808             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5809             \@@_qpoint:n { row - #####1 }
5810             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5811             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5812             {
5813                 \pgfpathrectanglecorners
5814                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5815                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5816             }
5817         }
5818     }
5819 }
5820 }

```

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`).

```
5821 \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

```
5822 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5823 {
5824   \bool_set_true:N \l_@@_nocolor_used_bool
5825   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5826   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5827   \clist_map_inline:Nn \l_@@_rows_tl
5828   {
5829     \clist_map_inline:Nn \l_@@_cols_tl
5830     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5831   }
5832 }
```

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`.

```
5833 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5834 {
5835   \clist_set_eq:NN \l_tmpa_clist #1
5836   \clist_clear:N #1
5837   \clist_map_inline:Nn \l_tmpa_clist
5838   {
5839     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5840     \tl_if_in:NnTF \l_tmpa_tl { - }
5841     { \@@_cut_on_hyphen:w ##1 \q_stop }
5842     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5843     \bool_lazy_or:nnT
5844     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5845     { \tl_if_blank_p:o \l_tmpa_tl }
5846     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5847     \bool_lazy_or:nnT
5848     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5849     { \tl_if_blank_p:o \l_tmpb_tl }
5850     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5851     \int_compare:nNnT \l_tmpb_tl > #2
5852     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5853     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5854     { \clist_put_right:Nn #1 { ####1 } }
5855   }
5856 }
```

The following command will be linked to `\cellcolor` in the tabular.

```
5857 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5858 {
5859   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5860   {
```

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`).

```
5861     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5862     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5863   }
5864   \ignorespaces
5865 }
```

The following command will be linked to `\rowcolor` in the tabular.

```

5866 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5867 {
5868   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5869   {
5870     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5871     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5872     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5873   }
5874   \ignorespaces
5875 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5876 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5877 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```

5878 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5879 {
5880   \peek_remove_spaces:n
5881   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5882 }

```

```

5883 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5884 {

```

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`.

```

5885   \seq_gclear:N \g_tmpa_seq
5886   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5887   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5888   \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`.

```

5889   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5890   {
5891     { \int_use:N \c@iRow }
5892     { \exp_not:n { #1 } }
5893     { \exp_not:n { #2 } }
5894     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5895   }
5896 }

```

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`).

```

5897 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5898 {
5899   \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).

```

5900     { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5901     {
5902       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5903       {
5904         \@@_rowlistcolors
5905         [ \exp_not:n { #2 } ]
5906         { #1 - \int_eval:n { \c@iRow - 1 } }
5907         { \exp_not:n { #3 } }
5908         [ \exp_not:n { #4 } ]
5909       }
5910     }
5911 }

```

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.

```

5912 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5913 {
5914   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5915   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5916   \seq_gclear:N \g_@@_rowlistcolors_seq
5917 }

5918 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5919 {
5920   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5921   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5922 }

```

The first mandatory argument of the command `\@@_rowlistcolors` which is writtent in the pre-`\CodeBefore` is of the form `i`: it means that the command must be applied to all the rows from the row `i` until the end of the tabular.

```

5923 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5924 {

```

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).

```

5925   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5926   {

```

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).

```

5927     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5928     {
5929       \exp_not:N \columncolor [ #1 ]
5930       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5931     }
5932   }
5933 }

5934 \hook_gput_code:nnn { begindocument } { . }
5935 {
5936   \IfPackageLoadedTF { colortbl }
5937   {
5938     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5939     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5940     \cs_new_protected:Npn \@@_revert_colortbl:

```

```

5941     {
5942         \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5943         {
5944             \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5945             \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5946         }
5947     }
5948 }
5949 { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5950 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

5951 \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`.

```

5952 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5953 {
5954     \int_if_zero:nTF \l_@@_first_col_int
5955     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5956     {
5957         \int_if_zero:nTF \c@jCol
5958         {
5959             \int_compare:nNnF \c@iRow = { -1 }
5960             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5961         }
5962         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5963     }
5964 }

```

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.

```

5965 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5966 {
5967     \int_if_zero:nF \c@iRow
5968     {
5969         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5970         {
5971             \int_compare:nNnT \c@jCol > \c_zero_int
5972             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5973         }
5974     }
5975 }

```

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`).

EmptyCol

The command `\EmptyCol` should be used in the preamble of an environment with preamble in a construction `>{...}` prefixing a column. No rule will be drawn in that column and no cell will be colored (whatever the coloring instructions used).

```

5976 \cs_new_protected:Npn \@@_EmptyCol:
5977 {
5978   \tl_set:Nx \l_tmpa_tl
5979   { { -2 } { \int_use:N \c@jCol } { 100 } { \int_use:N \c@jCol } { } }
5980   \seq_if_in:NoF \g__nicematrix_pos_of_blocks_seq \l_tmpa_tl
5981   { \seq_gput_right:No \g__nicematrix_pos_of_blocks_seq \l_tmpa_tl }
5982   \cellcolor { nocolor }
5983 }

```

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

5984 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
5985 {
5986   \IfPackageLoadedTF { tikz }
5987   {
5988     \IfPackageLoadedTF { booktabs }
5989     { #2 }
5990     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
5991   }
5992   { \@@_error:nn { TopRule~without~tikz } { #1 } }
5993 }
5994 \NewExpandableDocumentCommand { \@@_TopRule } { }
5995 { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }
5996 \cs_new:Npn \@@_TopRule_i:
5997 {
5998   \noalign \bgroup
5999   \peek_meaning:NTF [
6000   { \@@_TopRule_ii: }
6001   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6002 }
6003 \NewDocumentCommand \@@_TopRule_ii: { o }
6004 {
6005   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6006   {
6007     \@@_hline:n
6008     {
6009       position = \int_eval:n { \c@iRow + 1 } ,
6010       tikz =
6011       {
6012         line-width = #1 ,
6013         yshift = 0.25 \arrayrulewidth ,
6014         shorten-< = - 0.5 \arrayrulewidth
6015       } ,
6016       total-width = #1
6017     }
6018   }
6019   \skip_vertical:n { \belowrulesep + #1 }
6020   \egroup
6021 }
6022 \NewExpandableDocumentCommand { \@@_BottomRule } { }
6023 { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }
6024 \cs_new:Npn \@@_BottomRule_i:
6025 {
6026   \noalign \bgroup
6027   \peek_meaning:NTF [
6028   { \@@_BottomRule_ii: }
6029   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6030 }

```

```

6031 \NewDocumentCommand \@@_BottomRule_ii: { o }
6032 {
6033   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6034   {
6035     \@@_hline:n
6036     {
6037       position = \int_eval:n { \c@iRow + 1 } ,
6038       tikz =
6039       {
6040         line-width = #1 ,
6041         yshift = 0.25 \arrayrulewidth ,
6042         shorten~< = - 0.5 \arrayrulewidth
6043       } ,
6044       total-width = #1 ,
6045     }
6046   }
6047   \skip_vertical:N \aboverulesep
6048   \@@_create_row_node_i:
6049   \skip_vertical:n { #1 }
6050   \egroup
6051 }

6052 \NewExpandableDocumentCommand { \@@_MidRule } { } { }
6053 { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }

6054 \cs_new:Npn \@@_MidRule_i:
6055 {
6056   \noalign \bgroup
6057   \peek_meaning:NTF [
6058     { \@@_MidRule_ii: }
6059     { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6060   }

6061 \NewDocumentCommand \@@_MidRule_ii: { o }
6062 {
6063   \skip_vertical:N \aboverulesep
6064   \@@_create_row_node_i:
6065   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6066   {
6067     \@@_hline:n
6068     {
6069       position = \int_eval:n { \c@iRow + 1 } ,
6070       tikz =
6071       {
6072         line-width = #1 ,
6073         yshift = 0.25 \arrayrulewidth ,
6074         shorten~< = - 0.5 \arrayrulewidth
6075       } ,
6076       total-width = #1 ,
6077     }
6078   }
6079   \skip_vertical:n { \belowrulesep + #1 }
6080   \egroup
6081 }

```

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.

```

6082 \keys_define:nn { nicematrix / Rules }
6083 {

```



```

6084 position .int_set:N = \l_@@_position_int ,
6085 position .value_required:n = true ,
6086 start .int_set:N = \l_@@_start_int ,
6087 end .code:n =
6088   \bool_lazy_or:nnTF
6089     { \tl_if_empty_p:n { #1 } }
6090     { \str_if_eq_p:ee { #1 } { last } }
6091     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6092     { \int_set:Nn \l_@@_end_int { #1 } }
6093 }

```

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.

```

6094 \keys_define:nn { nicematrix / RulesBis }
6095 {
6096   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6097   multiplicity .initial:n = 1 ,
6098   dotted .bool_set:N = \l_@@_dotted_bool ,
6099   dotted .initial:n = false ,
6100   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]`.

```

6101   color .code:n =
6102     \@@_set_CT@arc@:n { #1 }
6103     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6104   color .value_required:n = true ,
6105   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6106   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.

```

6107   tikz .code:n =
6108     \IfPackageLoadedTF { tikz }
6109       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6110       { \@@_error:n { tikz~without~tikz } } ,
6111   tikz .value_required:n = true ,
6112   total-width .dim_set:N = \l_@@_rule_width_dim ,
6113   total-width .value_required:n = true ,
6114   width .meta:n = { total-width = #1 } ,
6115   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6116 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6117 \cs_new_protected:Npn \@@_vline:n #1
6118 {

```

The group is for the options.

```

6119   \group_begin:
6120   \int_set_eq:NN \l_@@_end_int \c@iRow
6121   \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).

```

6122     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6123     \@@_vline_i:
6124     \group_end:
6125 }

6126 \cs_new_protected:Npn \@@_vline_i:
6127 {

```

`\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`.

```

6128     \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6129     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6130     \l_tmpa_tl
6131     {

```

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.

```

6132     \bool_gset_true:N \g_tmpa_bool

6133     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6134     { \@@_test_vline_in_block:nnnnn ##1 }
6135     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6136     { \@@_test_vline_in_block:nnnnn ##1 }
6137     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6138     { \@@_test_vline_in_stroken_block:nnnn ##1 }
6139     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6140     \bool_if:NTF \g_tmpa_bool
6141     {
6142         \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.

```

6143         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6144     }
6145     {
6146         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6147         {
6148             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6149             \@@_vline_ii:
6150             \int_zero:N \l_@@_local_start_int
6151         }
6152     }
6153 }
6154 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6155 {
6156     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6157     \@@_vline_ii:
6158 }
6159 }

```

```

6160 \cs_new_protected:Npn \@@_test_in_corner_v:
6161 {
6162     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6163     {
6164         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6165         { \bool_set_false:N \g_tmpa_bool }
6166     }
6167     {
6168         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6169         {

```

```

6170         \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6171         { \bool_set_false:N \g_tmpa_bool }
6172         {
6173             \@@_if_in_corner:nT
6174             { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6175             { \bool_set_false:N \g_tmpa_bool }
6176         }
6177     }
6178 }
6179 }

```

```

6180 \cs_new_protected:Npn \@@_vline_ii:
6181 {
6182     \tl_clear:N \l_@@_tikz_rule_tl
6183     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6184     \bool_if:NTF \l_@@_dotted_bool
6185     \@@_vline_iv:
6186     {
6187         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6188         \@@_vline_iii:
6189         \@@_vline_v:
6190     }
6191 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6192 \cs_new_protected:Npn \@@_vline_iii:
6193 {
6194     \pgfpicture
6195     \pgfrememberpicturepositiononpagetrue
6196     \pgf@relevantforpicturesizefalse
6197     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6198     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6199     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6200     \dim_set:Nn \l_tmpb_dim
6201     {
6202         \pgf@x
6203         - 0.5 \l_@@_rule_width_dim
6204         +
6205         ( \arrayrulewidth * \l_@@_multiplicity_int
6206           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6207     }
6208     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6209     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6210     \bool_lazy_all:nT
6211     {
6212         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6213         { \cs_if_exist_p:N \CT@drsc@ }
6214         { ! \tl_if_blank_p:o \CT@drsc@ }
6215     }
6216     {
6217         \group_begin:
6218         \CT@drsc@
6219         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6220         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6221         \dim_set:Nn \l_@@_tmpd_dim
6222         {
6223             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6224             * ( \l_@@_multiplicity_int - 1 )
6225         }
6226         \pgfpathrectanglecorners
6227         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6228         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }

```

```

6229     \pgfusepath { fill }
6230     \group_end:
6231 }
6232 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6233 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6234 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6235 {
6236     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6237     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6238     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6239     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6240 }
6241 \CT@arc@
6242 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6243 \pgfsetrectcap
6244 \pgfusepathqstroke
6245 \endpgfpicture
6246 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6247 \cs_new_protected:Npn \@@_vline_iv:
6248 {
6249     \pgfpicture
6250     \pgfrememberpicturepositiononpagetrue
6251     \pgf@relevantforpicturesizefalse
6252     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6253     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6254     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6255     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6256     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6257     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6258     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6259     \CT@arc@
6260     \@@_draw_line:
6261     \endpgfpicture
6262 }

```

The following code is for the case when the user uses the key `tikz`.

```

6263 \cs_new_protected:Npn \@@_vline_v:
6264 {
6265     \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).

```

6266     \CT@arc@
6267     \tl_if_empty:NF \l_@@_rule_color_tl
6268     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6269     \pgfrememberpicturepositiononpagetrue
6270     \pgf@relevantforpicturesizefalse
6271     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6272     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6273     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6274     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6275     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6276     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6277     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6278     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6279     ( \l_tmpb_dim , \l_tmpa_dim ) --
6280     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6281     \end {tikzpicture }
6282 }

```

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).

```

6283 \cs_new_protected:Npn \@@_draw_vlines:
6284 {
6285   \int_step_inline:nnn
6286     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6287     {
6288       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6289       \c@jCol
6290       { \int_eval:n { \c@jCol + 1 } }
6291     }
6292     {
6293       \str_if_eq:eeF \l_@@_vlines_clist { all }
6294       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6295       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6296     }
6297 }

```

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}`.

```

6298 \cs_new_protected:Npn \@@_hline:n #1
6299 {

```

The group is for the options.

```

6300   \group_begin:
6301   \int_zero_new:N \l_@@_end_int
6302   \int_set_eq:NN \l_@@_end_int \c@jCol
6303   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6304   \@@_hline_i:
6305   \group_end:
6306 }
6307 \cs_new_protected:Npn \@@_hline_i:
6308 {
6309   \int_zero_new:N \l_@@_local_start_int
6310   \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`.

```

6311   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6312   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6313     \l_tmpb_tl
6314     {

```

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.

```

6315       \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6316       \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6317         { \@@_test_hline_in_block:nnnnn ##1 }
6318       \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6319         { \@@_test_hline_in_block:nnnnn ##1 }
6320       \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6321         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6322       \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6323       \bool_if:NTF \g_tmpa_bool
6324         {
6325           \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.

```

6326         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6327     }
6328     {
6329         \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6330         {
6331             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6332             \@@_hline_ii:
6333             \int_zero:N \l_@@_local_start_int
6334         }
6335     }
6336 }
6337 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6338 {
6339     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6340     \@@_hline_ii:
6341 }
6342 }

6343 \cs_new_protected:Npn \@@_test_in_corner_h:
6344 {
6345     \int_compare:nNtTF \l_tmpa_tl = { \c_iRow + 1 }
6346     {
6347         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6348         { \bool_set_false:N \g_tmpa_bool }
6349     }
6350     {
6351         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6352         {
6353             \int_compare:nNtTF \l_tmpa_tl = \c_one_int
6354             { \bool_set_false:N \g_tmpa_bool }
6355             {
6356                 \@@_if_in_corner:nT
6357                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6358                 { \bool_set_false:N \g_tmpa_bool }
6359             }
6360         }
6361     }
6362 }

6363 \cs_new_protected:Npn \@@_hline_ii:
6364 {
6365     \tl_clear:N \l_@@_tikz_rule_tl
6366     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6367     \bool_if:NTF \l_@@_dotted_bool
6368     \@@_hline_iv:
6369     {
6370         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6371         \@@_hline_iii:
6372         \@@_hline_v:
6373     }
6374 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6375 \cs_new_protected:Npn \@@_hline_iii:
6376 {
6377     \pgfpicture
6378     \pgfrememberpicturepositiononpagetrue
6379     \pgf@relevantforpicturesizefalse
6380     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }

```

```

6381 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6382 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6383 \dim_set:Nn \l_tmpb_dim
6384 {
6385   \pgf@y
6386   - 0.5 \l_@@_rule_width_dim
6387   +
6388   ( \arrayrulewidth * \l_@@_multiplicity_int
6389     + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6390 }
6391 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6392 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6393 \bool_lazy_all:nT
6394 {
6395   { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6396   { \cs_if_exist_p:N \CT@drsc@ }
6397   { ! \tl_if_blank_p:o \CT@drsc@ }
6398 }
6399 {
6400   \group_begin:
6401   \CT@drsc@
6402   \dim_set:Nn \l_@@_tmpd_dim
6403   {
6404     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6405     * ( \l_@@_multiplicity_int - 1 )
6406   }
6407   \pgfpathrectanglecorners
6408   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6409   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6410   \pgfusepathqfill
6411   \group_end:
6412 }
6413 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6414 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6415 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6416 {
6417   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6418   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6419   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6420   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6421 }
6422 \CT@arc@
6423 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6424 \pgfsetrectcap
6425 \pgfusepathqstroke
6426 \endpgfpicture
6427 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6428 \cs_new_protected:Npn \@@_hline_iv:
6429 {
6430   \pgfpicture
6431   \pgfrememberpicturepositiononpagetrue
6432   \pgf@relevantforpicturesizefalse
6433   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6434   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6435   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6436   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6437   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6438   \int_compare:nNtT \l_@@_local_start_int = \c_one_int
6439   {
6440     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6441     \bool_if:NF \g_@@_delims_bool
6442     { \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.

```

6443   \tl_if_eq:NnF \g_@@_left_delim_tl (
6444     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6445   )
6446   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6447   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6448   \int_compare:nNtT \l_@@_local_end_int = \c_jCol
6449   {
6450     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6451     \bool_if:NF \g_@@_delims_bool
6452     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6453     \tl_if_eq:NnF \g_@@_right_delim_tl )
6454     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6455   }
6456   \CT@arc@
6457   \@@_draw_line:
6458   \endpgfpicture
6459 }

```

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`).

```

6460 \cs_new_protected:Npn \@@_hline_v:
6461 {
6462   \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).

```

6463   \CT@arc@
6464   \tl_if_empty:NF \l_@@_rule_color_tl
6465   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6466   \pgfrememberpicturepositiononpagetrue
6467   \pgf@relevantforpicturesizefalse
6468   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6469   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6470   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6471   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6472   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6473   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```



```

6474 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6475 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6476 ( \l_tmpa_dim , \l_tmpb_dim ) --
6477 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6478 \end { tikzpicture }
6479 }

```

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).

```

6480 \cs_new_protected:Npn \@@_draw_hlines:
6481 {
6482   \int_step_inline:nnn
6483   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6484   {
6485     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6486     \c@iRow
6487     { \int_eval:n { \c@iRow + 1 } }
6488   }
6489   {
6490     \str_if_eq:eeF \l_@@_hlines_clist { all }
6491     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6492     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6493   }
6494 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6495 \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.

```

6496 \cs_set:Npn \@@_Hline_i:n #1
6497 {
6498   \peek_remove_spaces:n
6499   {
6500     \peek_meaning:NTF \Hline
6501     { \@@_Hline_ii:nn { #1 + 1 } }
6502     { \@@_Hline_iii:n { #1 } }
6503   }
6504 }
6505 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6506 \cs_set:Npn \@@_Hline_iii:n #1
6507 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6508 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6509 {
6510   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6511   \skip_vertical:N \l_@@_rule_width_dim
6512   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6513   {
6514     \@@_hline:n
6515     {
6516       multiplicity = #1 ,
6517       position = \int_eval:n { \c@iRow + 1 } ,
6518       total-width = \dim_use:N \l_@@_rule_width_dim ,
6519       #2
6520     }
6521   }
6522   \egroup
6523 }

```

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`).

```

6524 \cs_new_protected:Npn \@@_custom_line:n #1
6525 {
6526   \str_clear_new:N \l_@@_command_str
6527   \str_clear_new:N \l_@@_ccommand_str
6528   \str_clear_new:N \l_@@_letter_str
6529   \tl_clear_new:N \l_@@_other_keys_tl
6530   \keys_set:known:nN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6531   \bool_lazy_all:nTF
6532   {
6533     { \str_if_empty_p:N \l_@@_letter_str }
6534     { \str_if_empty_p:N \l_@@_command_str }
6535     { \str_if_empty_p:N \l_@@_ccommand_str }
6536   }
6537   { \@@_error:n { No~letter-and-no~command } }
6538   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6539 }
6540 \keys_define:nn { nicematrix / custom-line }
6541 {
6542   letter .str_set:N = \l_@@_letter_str ,
6543   letter .value_required:n = true ,
6544   command .str_set:N = \l_@@_command_str ,
6545   command .value_required:n = true ,
6546   ccommand .str_set:N = \l_@@_ccommand_str ,
6547   ccommand .value_required:n = true ,
6548 }

```

```

6549 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6550 \cs_new_protected:Npn \@@_custom_line_i:n #1
6551 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6552   \bool_set_false:N \l_@@_tikz_rule_bool
6553   \bool_set_false:N \l_@@_dotted_rule_bool
6554   \bool_set_false:N \l_@@_color_bool
6555   \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6556   \bool_if:NT \l_@@_tikz_rule_bool
6557   {
6558     \IfPackageLoadedF { tikz }
6559     { \@@_error:n { tikz-in-custom-line-without-tikz } }
6560     \bool_if:NT \l_@@_color_bool
6561     { \@@_error:n { color-in-custom-line-with-tikz } }
6562   }
6563   \bool_if:NT \l_@@_dotted_rule_bool
6564   {
6565     \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6566     { \@@_error:n { key-multiplicity-with-dotted } }
6567   }
6568   \str_if_empty:NF \l_@@_letter_str
6569   {

```

```

6570 \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6571 { \@@_error:n { Several~letters } }
6572 {
6573   \tl_if_in:NoTF
6574   \c_@@_forbidden_letters_str
6575   \l_@@_letter_str
6576   { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6577   {

```

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.

```

6578 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6579 { \@@_v_custom_line:n { #1 } }
6580 }
6581 }
6582 }
6583 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6584 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6585 }

6586 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6587 \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.

```

6588 \keys_define:nn { nicematrix / custom-line-bis }
6589 {
6590   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6591   multiplicity .initial:n = 1 ,
6592   multiplicity .value_required:n = true ,
6593   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6594   color .value_required:n = true ,
6595   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6596   tikz .value_required:n = true ,
6597   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6598   dotted .value_forbidden:n = true ,
6599   total-width .code:n = { } ,
6600   total-width .value_required:n = true ,
6601   width .code:n = { } ,
6602   width .value_required:n = true ,
6603   sep-color .code:n = { } ,
6604   sep-color .value_required:n = true ,
6605   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6606 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6607 \bool_new:N \l_@@_dotted_rule_bool
6608 \bool_new:N \l_@@_tikz_rule_bool
6609 \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`.

```

6610 \keys_define:nn { nicematrix / custom-line-width }
6611 {
6612   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6613   multiplicity .initial:n = 1 ,
6614   multiplicity .value_required:n = true ,
6615   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6616   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }

```

```

6617         \bool_set_true:N \l_@@_total_width_bool ,
6618     total-width .value_required:n = true ,
6619     width .meta:n = { total-width = #1 } ,
6620     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6621 }

```

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).

```

6622 \cs_new_protected:Npn \@@_h_custom_line:n #1
6623 {

```

We use \cs_set:nopar: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).

```

6624     \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6625     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6626 }

```

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).

```

6627 \cs_new_protected:Npn \@@_c_custom_line:n #1
6628 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6629     \exp_args:Nc \NewExpandableDocumentCommand
6630     { nicematrix - \l_@@_ccommand_str }
6631     { 0 { } m }
6632     {
6633         \noalign
6634         {
6635             \@@_compute_rule_width:n { #1 , ##1 }
6636             \skip_vertical:n { \l_@@_rule_width_dim }
6637             \clist_map_inline:nn
6638             { ##2 }
6639             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6640         }
6641     }
6642     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6643 }

```

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*.

```

6644 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6645 {
6646     \tl_if_in:nnTF { #2 } { - }
6647     { \@@_cut_on_hyphen:w #2 \q_stop }
6648     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6649     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6650     {
6651         \@@_hline:n
6652         {
6653             #1 ,
6654             start = \l_tmpa_tl ,
6655             end = \l_tmpb_tl ,
6656             position = \int_eval:n { \c@iRow + 1 } ,
6657             total-width = \dim_use:N \l_@@_rule_width_dim
6658         }
6659     }
6660 }

```

```

6661 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6662 {
6663   \bool_set_false:N \l_@@_tikz_rule_bool
6664   \bool_set_false:N \l_@@_total_width_bool
6665   \bool_set_false:N \l_@@_dotted_rule_bool
6666   \keys_set_known:n { nicematrix / custom-line-width } { #1 }
6667   \bool_if:NF \l_@@_total_width_bool
6668   {
6669     \bool_if:NTF \l_@@_dotted_rule_bool
6670     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6671     {
6672       \bool_if:NF \l_@@_tikz_rule_bool
6673       {
6674         \dim_set:Nn \l_@@_rule_width_dim
6675         {
6676           \arrayrulewidth * \l_@@_multiplicity_int
6677           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6678         }
6679       }
6680     }
6681   }
6682 }
6683 \cs_new_protected:Npn \@@_v_custom_line:n #1
6684 {
6685   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6686   \tl_gput_right:Ne \g_@@_array_preamble_tl
6687   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6688   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6689   {
6690     \@@_vline:n
6691     {
6692       #1 ,
6693       position = \int_eval:n { \c@jCol + 1 } ,
6694       total-width = \dim_use:N \l_@@_rule_width_dim
6695     }
6696   }
6697   \@@_rec_preamble:n
6698 }
6699 \@@_custom_line:n
6700 { 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.

```

6701 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6702 {
6703   \int_compare:nNnT \l_tmpa_tl > { #1 }
6704   {
6705     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6706     {
6707       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6708       {
6709         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6710         { \bool_gset_false:N \g_tmpa_bool }
6711       }
6712     }
6713   }
6714 }

```

The same for vertical rules.

```

6715 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6716 {
6717   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6718   {
6719     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6720     {
6721       \int_compare:nNnT \l_tmpb_tl > { #2 }
6722       {
6723         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6724         { \bool_gset_false:N \g_tmpa_bool }
6725       }
6726     }
6727   }
6728 }

6729 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6730 {
6731   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6732   {
6733     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6734     {
6735       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6736       { \bool_gset_false:N \g_tmpa_bool }
6737       {
6738         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6739         { \bool_gset_false:N \g_tmpa_bool }
6740       }
6741     }
6742   }
6743 }

6744 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6745 {
6746   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6747   {
6748     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6749     {
6750       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6751       { \bool_gset_false:N \g_tmpa_bool }
6752       {
6753         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6754         { \bool_gset_false:N \g_tmpa_bool }
6755       }
6756     }
6757   }
6758 }

```

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.

```

6759 \cs_new_protected:Npn \@@_compute_corners:
6760 {
6761   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6762   { \@@_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`).

```

6763 \clist_clear:N \l_@@_corners_cells_clist
6764 \clist_map_inline:Nn \l_@@_corners_cells_clist
6765 {
6766   \str_case:nnF { ##1 }
6767   {
6768     { NW }
6769     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6770     { NE }
6771     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6772     { SW }
6773     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6774     { SE }
6775     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6776   }
6777   { \@@_error:nn { bad~corner } { ##1 } }
6778 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6779 \clist_if_empty:NF \l_@@_corners_cells_clist
6780 {

```

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.

```

6781 \tl_gput_right:Ne \g_@@_aux_tl
6782 {
6783   \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6784   { \l_@@_corners_cells_clist }
6785 }
6786 }
6787 }

```

```

6788 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6789 {
6790   \int_step_inline:nnn { #1 } { #3 }
6791   {
6792     \int_step_inline:nnn { #2 } { #4 }
6793     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6794   }
6795 }

```

```

6796 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6797 {
6798   \cs_if_exist:cTF
6799   { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6800   \prg_return_true:
6801   \prg_return_false:
6802 }

```

“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.

```
6803 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6804 {
```

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.

```
6805 \bool_set_false:N \l_tmpa_bool
6806 \int_zero_new:N \l_@@_last_empty_row_int
6807 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6808 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6809 {
6810 \bool_lazy_or:nnTF
6811 {
6812 \cs_if_exist_p:c
6813 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6814 }
6815 { \@@_if_in_block_p:nn { ##1 } { #2 } }
6816 { \bool_set_true:N \l_tmpa_bool }
6817 {
6818 \bool_if:NF \l_tmpa_bool
6819 { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6820 }
6821 }
```

Now, you determine the last empty cell in the row of number 1.

```
6822 \bool_set_false:N \l_tmpa_bool
6823 \int_zero_new:N \l_@@_last_empty_column_int
6824 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6825 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6826 {
6827 \bool_lazy_or:nnTF
6828 {
6829 \cs_if_exist_p:c
6830 { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6831 }
6832 { \@@_if_in_block_p:nn { #1 } { ##1 } }
6833 { \bool_set_true:N \l_tmpa_bool }
6834 {
6835 \bool_if:NF \l_tmpa_bool
6836 { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6837 }
6838 }
```

Now, we loop over the rows.

```
6839 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6840 {
```

We treat the row number `##1` with another loop.

```
6841 \bool_set_false:N \l_tmpa_bool
6842 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6843 {
6844 \bool_lazy_or:nnTF
6845 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6846 { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6847 { \bool_set_true:N \l_tmpa_bool }
6848 {
6849 \bool_if:NF \l_tmpa_bool
6850 {
6851 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6852 \clist_put_right:Nn
6853 \l_@@_corners_cells_clist
```



```

6854             { ##1 - #####1 }
6855         \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6856     }
6857 }
6858 }
6859 }
6860 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6861 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6862 \cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }

```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient:
`\clist_if_in:NcT \l_@@_corners_cells_clist { #1 } ...`

24 The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```

6863 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6864 \keys_define:nn { nicematrix / NiceMatrixBlock }
6865 {
6866     auto-columns-width .code:n =
6867     {
6868         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6869         \dim_gzero_new:N \g_@@_max_cell_width_dim
6870         \bool_set_true:N \l_@@_auto_columns_width_bool
6871     }
6872 }

6873 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6874 {
6875     \int_gincr:N \g_@@_NiceMatrixBlock_int
6876     \dim_zero:N \l_@@_columns_width_dim
6877     \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6878     \bool_if:NT \l_@@_block_auto_columns_width_bool
6879     {
6880         \cs_if_exist:cT
6881         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6882         {
6883             \dim_set:Nn \l_@@_columns_width_dim
6884             {
6885                 \use:c
6886                 { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6887             }
6888         }
6889     }
6890 }

```

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`).

```

6891 {
6892     \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.

```

6893     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6894     {
6895         \bool_if:NT \l_@@_block_auto_columns_width_bool
6896         {
6897             \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6898             \iow_shipout:Ne \@mainaux
6899             {
6900                 \cs_gset:cpn
6901                 { @@ _ 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.

```

6902         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6903     }
6904     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6905 }
6906 }
6907 \ignorespacesafterend
6908 }

```

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).

```

6909 \cs_new_protected:Npn \@@_create_extra_nodes:
6910 {
6911     \bool_if:nTF \l_@@_medium_nodes_bool
6912     {
6913         \bool_if:NTF \l_@@_large_nodes_bool
6914         \@@_create_medium_and_large_nodes:
6915         \@@_create_medium_nodes:
6916     }
6917     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6918 }

```

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`.

```

6919 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6920 {
6921     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:

```

```

6922 {
6923   \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6924   \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6925   \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6926   \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6927 }
6928 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6929 {
6930   \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6931   \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6932   \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6933   \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6934 }

```

We begin the two nested loops over the rows and the columns of the array.

```

6935 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6936 {
6937   \int_step_variable:nnNn
6938   \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.

```

6939 {
6940   \cs_if_exist:cT
6941   { 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.

```

6942 {
6943   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6944   \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6945   { \dim_min:vn { l_@@_row\_@@_i: _min_dim } \pgf@y }
6946   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6947   {
6948     \dim_set:cn { l_@@_column\_@@_j: _min_dim }
6949     { \dim_min:vn { l_@@_column\_@@_j: _min_dim } \pgf@x }
6950   }

```

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.

```

6951   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6952   \dim_set:cn { l_@@_row\_@@_i: _max_dim }
6953   { \dim_max:vn { l_@@_row\_@@_i: _max_dim } \pgf@y }
6954   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6955   {
6956     \dim_set:cn { l_@@_column\_@@_j: _max_dim }
6957     { \dim_max:vn { l_@@_column\_@@_j: _max_dim } \pgf@x }
6958   }
6959 }
6960 }
6961 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6962 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6963 {
6964   \dim_compare:nNnT
6965   { \dim_use:c { l_@@_row\_@@_i: _min _ dim } } = \c_max_dim
6966   {
6967     \@@_qpoint:n { row - \@@_i: - base }
6968     \dim_set:cn { l_@@_row\_@@_i: _max _ dim } \pgf@y
6969     \dim_set:cn { l_@@_row\_@@_i: _min _ dim } \pgf@y
6970   }
6971 }
6972 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

```

6973 {
6974   \dim_compare:nNnT
6975     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6976     {
6977       \@@_qpoint:n { col - \@@_j: }
6978       \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6979       \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6980     }
6981   }
6982 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6983 \cs_new_protected:Npn \@@_create_medium_nodes:
6984 {
6985   \pgfpicture
6986   \pgfrememberpicturepositiononpagetrue
6987   \pgf@relevantforpicturesizefalse
6988   \@@_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”.

```

6989   \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6990   \@@_create_nodes:
6991   \endpgfpicture
6992 }

```

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:`.

```

6993 \cs_new_protected:Npn \@@_create_large_nodes:
6994 {
6995   \pgfpicture
6996   \pgfrememberpicturepositiononpagetrue
6997   \pgf@relevantforpicturesizefalse
6998   \@@_computations_for_medium_nodes:
6999   \@@_computations_for_large_nodes:
7000   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7001   \@@_create_nodes:
7002   \endpgfpicture
7003 }
7004 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7005 {
7006   \pgfpicture
7007   \pgfrememberpicturepositiononpagetrue
7008   \pgf@relevantforpicturesizefalse
7009   \@@_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”.

```

7010   \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
7011   \@@_create_nodes:
7012   \@@_computations_for_large_nodes:
7013   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
7014   \@@_create_nodes:
7015   \endpgfpicture
7016 }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

For “large nodes”, the exterior rows and columns don’t interfer. 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.

```

7017 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7018 {
7019   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7020   \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`.

```

7021   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7022   {
7023     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7024     {
7025       (
7026         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7027         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7028       )
7029       / 2
7030     }
7031     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7032     { l_@@_row _ \@@_i: _ min _ dim }
7033   }
7034   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7035   {
7036     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7037     {
7038       (
7039         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7040         \dim_use:c
7041           { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7042       )
7043       / 2
7044     }
7045     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7046     { l_@@_column _ \@@_j: _ max _ dim }
7047   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7048   \dim_sub:cn
7049     { l_@@_column _ 1 _ min _ dim }
7050     \l_@@_left_margin_dim
7051   \dim_add:cn
7052     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7053     \l_@@_right_margin_dim
7054 }

```

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).

```

7055 \cs_new_protected:Npn \@@_create_nodes:
7056 {
7057   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7058   {
7059     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7060     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7061       \@@_pgf_rect_node:nnnnn
7062       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7063       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

7064         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7065         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7066         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7067     \str_if_empty:NF \l_@@_name_str
7068     {
7069         \pgfnodealias
7070         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7071         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7072     }
7073 }
7074 }

```

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 .

```

7075     \seq_map_pairwise_function:NNN
7076     \g_@@_multicolumn_cells_seq
7077     \g_@@_multicolumn_sizes_seq
7078     \@@_node_for_multicolumn:nn
7079 }

7080 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7081 {
7082     \cs_set_nopar:Npn \@@_i: { #1 }
7083     \cs_set_nopar:Npn \@@_j: { #2 }
7084 }

```

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”).

```

7085 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7086 {
7087     \@@_extract_coords_values: #1 \q_stop
7088     \@@_pgf_rect_node:nnnnn
7089     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7090     { \dim_use:c { l_@@_column_ \@@_j: _min _ dim } }
7091     { \dim_use:c { l_@@_row_ \@@_i: _min _ dim } }
7092     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7093     { \dim_use:c { l_@@_row_ \@@_i: _ max _ dim } }
7094     \str_if_empty:NF \l_@@_name_str
7095     {
7096         \pgfnodealias
7097         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7098         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7099     }
7100 }

```

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).

```

7101 \keys_define:nn { nicematrix / Block / FirstPass }
7102 {
7103     j .code:n = \str_set:Nn \l_@@_hpos_block_str j

```

```

7104         \bool_set_true:N \l_@@_p_block_bool ,
7105     j .value_forbidden:n = true ,
7106     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7107     l .value_forbidden:n = true ,
7108     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7109     r .value_forbidden:n = true ,
7110     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7111     c .value_forbidden:n = true ,
7112     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7113     L .value_forbidden:n = true ,
7114     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7115     R .value_forbidden:n = true ,
7116     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7117     C .value_forbidden:n = true ,
7118     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7119     t .value_forbidden:n = true ,
7120     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7121     T .value_forbidden:n = true ,
7122     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7123     b .value_forbidden:n = true ,
7124     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7125     B .value_forbidden:n = true ,
7126     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7127     m .value_forbidden:n = true ,
7128     v-center .meta:n = m ,
7129     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7130     p .value_forbidden:n = true ,
7131     color .code:n =
7132         \@@_color:n { #1 }
7133         \tl_set_rescan:Nnn
7134             \l_@@_draw_tl
7135             { \char_set_catcode_other:N ! }
7136             { #1 } ,
7137     color .value_required:n = true ,
7138     respect-arraystretch .code:n =
7139         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7140     respect-arraystretch .value_forbidden:n = true ,
7141 }

```

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.

```

7142 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7143 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7144 {

```

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).

```

7145     \peek_remove_spaces:n
7146     {
7147         \tl_if_blank:nTF { #2 }
7148             { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7149             {
7150                 \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7151                 \@@_Block_i_czech \@@_Block_i
7152                 #2 \q_stop
7153             }
7154         { #1 } { #3 } { #4 }
7155     }
7156 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7157 \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`.

```
7158 {
7159   \char_set_catcode_active:N -
7160   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7161 }
```

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.

```
7162 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7163 {
```

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).

```
7164   \bool_lazy_or:nnTF
7165     { \tl_if_blank_p:n { #1 } }
7166     { \str_if_eq_p:ee { * } { #1 } }
7167     { \int_set:Nn \l_tmpa_int { 100 } }
7168     { \int_set:Nn \l_tmpa_int { #1 } }
7169   \bool_lazy_or:nnTF
7170     { \tl_if_blank_p:n { #2 } }
7171     { \str_if_eq_p:ee { * } { #2 } }
7172     { \int_set:Nn \l_tmpb_int { 100 } }
7173     { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7174   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7175     {
7176       \tl_if_empty:NTF \l_@@_hpos_cell_tl
7177         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7178         { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7179     }
7180     { \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.

```
7181   \keys_set:known:nn { nicematrix / Block / FirstPass } { #3 }
7182   \tl_set:Ne \l_tmpa_tl
7183   {
7184     { \int_use:N \c@iRow }
7185     { \int_use:N \c@jCol }
7186     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7187     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7188   }
```

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).

```
7189   \bool_set_false:N \l_tmpa_bool
7190   \bool_if:NT \l_@@_amp_in_blocks_bool
```


`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7191     { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7192     \bool_case:nF
7193     {
7194         \l_tmpa_bool                { \@@_Block_vii:eennn }
7195         \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.

```

7196         \l_@@_X_bool                { \@@_Block_v:eennn }
7197         { \tl_if_empty_p:n { #5 } }   { \@@_Block_v:eennn }
7198         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7199         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7200     }
7201     { \@@_Block_v:eennn }
7202     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7203 }

```

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.

```

7204 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7205 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7206 {
7207     \int_gincr:N \g_@@_block_box_int
7208     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7209     {
7210         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7211         {
7212             \@@_actually_diagbox:nnnnnn
7213             { \int_use:N \c@iRow }
7214             { \int_use:N \c@jCol }
7215             { \int_eval:n { \c@iRow + #1 - 1 } }
7216             { \int_eval:n { \c@jCol + #2 - 1 } }
7217             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7218             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7219         }
7220     }
7221     \box_gclear_new:c
7222     { g_@@_block_box_int _box _ \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7223     \hbox_gset:cn
7224     { g_@@_block_box_int _box _ \int_use:N \g_@@_block_box_int _box }
7225     {

```

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}`).

```

7226 \tl_if_empty:NTF \l_@@_color_tl
7227 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7228 { \@@_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`.

```

7229 \int_compare:nNnT { #1 } = \c_one_int
7230 {
7231   \int_if_zero:nTF \c@iRow
7232   {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7233 \cs_set_eq:NN \Block \@@_NullBlock:
7234 \l_@@_code_for_first_row_tl
7235 }
7236 {
7237   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7238   {
7239     \cs_set_eq:NN \Block \@@_NullBlock:
7240     \l_@@_code_for_last_row_tl
7241   }
7242 }
7243 \g_@@_row_style_tl
7244 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7245 \@@_reset_arraystretch:
7246 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7247 #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.).

```

7248 \@@_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}`.

```

7249 \bool_if:NTF \l_@@_tabular_bool

```

```

7250 {
7251   \bool_lazy_all:nTF
7252   {
7253     { \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.

```

7254     { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7255     { ! \g_@@_rotate_bool }
7256   }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7257 {
7258   \use:e
7259   {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7260     \exp_not:N \begin { minipage }%
7261     [ \str_lowercase:o \l_@@_vpos_block_str ]
7262     { \l_@@_col_width_dim }
7263     \str_case:on \l_@@_hpos_block_str
7264     { c \centering r \raggedleft l \raggedright }
7265   }
7266   #5
7267   \end { minipage }
7268 }

```

In the other cases, we use a `{tabular}`.

```

7269 {
7270   \bool_if:NT \c_@@_testphase_table_bool
7271   { \tagpdfsetup { table / tagging = presentation } }
7272   \use:e
7273   {
7274     \exp_not:N \begin { tabular }%
7275     [ \str_lowercase:o \l_@@_vpos_block_str ]
7276     { @ { } \l_@@_hpos_block_str @ { } }
7277   }
7278   #5
7279   \end { tabular }
7280 }
7281 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7282 {
7283   \c_math_toggle_token
7284   \use:e
7285   {
7286     \exp_not:N \begin { array }%
7287     [ \str_lowercase:o \l_@@_vpos_block_str ]
7288     { @ { } \l_@@_hpos_block_str @ { } }
7289   }
7290   #5
7291   \end { array }
7292   \c_math_toggle_token
7293 }
7294 }

```

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).

```

7295   \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.

```

7296     \int_compare:nNtT { #2 } = \c_one_int
7297     {
7298         \dim_gset:Nn \g_@@_blocks_wd_dim
7299         {
7300             \dim_max:nn
7301             \g_@@_blocks_wd_dim
7302             {
7303                 \box_wd:c
7304                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7305             }
7306         }
7307     }

```

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.

```

7308     \bool_lazy_and:nnT
7309     { \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.

```

7310     { \str_if_empty_p:N \l_@@_vpos_block_str }
7311     {
7312         \dim_gset:Nn \g_@@_blocks_ht_dim
7313         {
7314             \dim_max:nn
7315             \g_@@_blocks_ht_dim
7316             {
7317                 \box_ht:c
7318                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7319             }
7320         }
7321         \dim_gset:Nn \g_@@_blocks_dp_dim
7322         {
7323             \dim_max:nn
7324             \g_@@_blocks_dp_dim
7325             {
7326                 \box_dp:c
7327                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7328             }
7329         }
7330     }
7331     \seq_gput_right:Ne \g_@@_blocks_seq
7332     {
7333         \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.

```

7334     {
7335         \exp_not:n { #3 } ,
7336         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7337     \bool_if:NT \g_@@_rotate_bool
7338     {
7339         \bool_if:NTF \g_@@_rotate_c_bool
7340         { m }
7341         { \int_compare:nNtT \c@iRow = \l_@@_last_row_int T }
7342     }
7343 }

```

```

7344     {
7345         \box_use_drop:c
7346         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7347     }
7348 }
7349 \bool_set_false:N \g_@@_rotate_c_bool
7350 }

7351 \cs_new:Npn \@@_adjust_hpos_rotate:
7352 {
7353     \bool_if:NT \g_@@_rotate_bool
7354     {
7355         \str_set:Ne \l_@@_hpos_block_str
7356         {
7357             \bool_if:NTF \g_@@_rotate_c_bool
7358             { c }
7359             {
7360                 \str_case:onF \l_@@_vpos_block_str
7361                 { b l B l t r T r }
7362                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7363             }
7364         }
7365     }
7366 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7367 \cs_new_protected:Npn \@@_rotate_box_of_block:
7368 {
7369     \box_grotate:cn
7370     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7371     { 90 }
7372     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7373     {
7374         \vbox_gset_top:cn
7375         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7376         {
7377             \skip_vertical:n { 0.8 ex }
7378             \box_use:c
7379             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7380         }
7381     }
7382     \bool_if:NT \g_@@_rotate_c_bool
7383     {
7384         \hbox_gset:cn
7385         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7386         {
7387             \c_math_toggle_token
7388             \vcenter
7389             {
7390                 \box_use:c
7391                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7392             }
7393             \c_math_toggle_token
7394         }
7395     }
7396 }

```

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:nnnnnn`).

#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 $\text{key}=\text{values}$ 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.

```

7397 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7398 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7399 {
7400   \seq_gput_right:Ne \g_@@_blocks_seq
7401   {
7402     \l_tmpa_tl
7403     { \exp_not:n { #3 } }
7404     {
7405       \bool_if:NTF \l_@@_tabular_bool
7406       {
7407         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7408       \@@_reset_arraystretch:
7409       \exp_not:n
7410       {
7411         \dim_zero:N \extrarowheight
7412         #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).

```

7413       \bool_if:NT \c_@@_testphase_table_bool
7414       { \tag_stop:n { table } }
7415       \use:e
7416       {
7417         \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7418         { @ { } \l_@@_hpos_block_str @ { } }
7419       }
7420       #5
7421       \end { tabular }
7422     }
7423   \group_end:
7424 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7425   {
7426     \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7427     \@@_reset_arraystretch:
7428     \exp_not:n
7429     {
7430       \dim_zero:N \extrarowheight
7431       #4
7432       \c_math_toggle_token
7433       \use:e
7434       {
7435         \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7436         { @ { } \l_@@_hpos_block_str @ { } }
7437       }
7438       #5
7439       \end { array }
7440       \c_math_toggle_token
7441     }
7442   \group_end:
7443 }
7444 }
7445 }
7446 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7447 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7448 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7449 {
7450   \seq_gput_right:Ne \g_@@_blocks_seq
7451   {
7452     \l_tmpa_tl
7453     { \exp_not:n { #3 } }
7454     {
7455       \group_begin:
7456       \exp_not:n { #4 #5 }
7457       \group_end:
7458     }
7459   }
7460 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7461 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7462 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7463 {
7464   \seq_gput_right:Ne \g_@@_blocks_seq
7465   {
7466     \l_tmpa_tl
7467     { \exp_not:n { #3 } }
7468     { \exp_not:n { #4 #5 } }
7469   }
7470 }

```

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).

```

7471 \keys_define:nn { nicematrix / Block / SecondPass }
7472 {
7473   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7474   ampersand-in-blocks .default:n = true ,
7475   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7476 tikz .code:n =
7477   \IfPackageLoadedTF { tikz }
7478   { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7479   { \@@_error:n { tikz~key~without~tikz } } ,
7480 tikz .value_required:n = true ,
7481 fill .code:n =
7482   \tl_set_rescan:Nnn
7483   \l_@@_fill_tl
7484   { \char_set_catcode_other:N ! }
7485   { #1 } ,
7486 fill .value_required:n = true ,
7487 opacity .tl_set:N = \l_@@_opacity_tl ,
7488 opacity .value_required:n = true ,
7489 draw .code:n =
7490   \tl_set_rescan:Nnn
7491   \l_@@_draw_tl
7492   { \char_set_catcode_other:N ! }
7493   { #1 } ,
7494 draw .default:n = default ,
7495 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7496 rounded-corners .default:n = 4 pt ,
7497 color .code:n =
7498   \@@_color:n { #1 }

```

```

7499 \tl_set_rescan:Nnn
7500 \l_@@_draw_tl
7501 { \char_set_catcode_other:N ! }
7502 { #1 } ,
7503 borders .clist_set:N = \l_@@_borders_clist ,
7504 borders .value_required:n = true ,
7505 hvlines .meta:n = { vlines , hlines } ,
7506 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7507 vlines .default:n = true ,
7508 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7509 hlines .default:n = true ,
7510 line-width .dim_set:N = \l_@@_line_width_dim ,
7511 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7512 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7513 \bool_set_true:N \l_@@_p_block_bool ,
7514 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7515 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7516 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7517 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7518 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7519 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7520 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7521 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7522 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7523 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7524 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7525 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7526 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7527 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7528 m .value_forbidden:n = true ,
7529 v-center .meta:n = m ,
7530 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7531 p .value_forbidden:n = true ,
7532 name .tl_set:N = \l_@@_block_name_str ,
7533 name .value_required:n = true ,
7534 name .initial:n = ,
7535 respect-arraystretch .code:n =
7536 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7537 respect-arraystretch .value_forbidden:n = true ,
7538 transparent .bool_set:N = \l_@@_transparent_bool ,
7539 transparent .default:n = true ,
7540 transparent .initial:n = false ,
7541 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7542 }

```

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.

```

7543 \cs_new_protected:Npn \@@_draw_blocks:
7544 {
7545 \bool_if:NTF \c_@@_recent_array_bool
7546 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7547 { \cs_set_eq:NN \ialign \@@_old_ialign: }
7548 \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7549 }
7550 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7551 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7552 \int_zero_new:N \l_@@_last_row_int
7553 \int_zero_new:N \l_@@_last_col_int

```


We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7554 \int_compare:nNnTF { #3 } > { 98 }
7555   { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7556   { \int_set:Nn \l_@@_last_row_int { #3 } }
7557 \int_compare:nNnTF { #4 } > { 98 }
7558   { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7559   { \int_set:Nn \l_@@_last_col_int { #4 } }
7560 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7561 {
7562   \bool_lazy_and:nnTF
7563     \l_@@_preamble_bool
7564     {
7565       \int_compare_p:n
7566         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7567     }
7568     {
7569       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7570       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7571       \@@_msg_redirect_name:nn { columns-not-used } { none }
7572     }
7573     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7574 }
7575 {
7576   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7577   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7578   {
7579     \@@_Block_v:nneenn
7580     { #1 }
7581     { #2 }
7582     { \int_use:N \l_@@_last_row_int }
7583     { \int_use:N \l_@@_last_col_int }
7584     { #5 }
7585     { #6 }
7586   }
7587 }
7588 }
```

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

```

7589 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7590 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7591 {
```

The group is for the keys.

```

7592 \group_begin:
7593 \int_compare:nNnT { #1 } = { #3 }
7594   { \str_set:Nn \l_@@_vpos_block_str { t } }
7595 \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 then `\str_if_in:nnT`.

```

7596 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7597 \bool_lazy_and:nnT
7598   \l_@@_vlines_block_bool
7599   { ! \l_@@_ampersand_bool }
7600 {
```

```

7601     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7602     {
7603         \@@_vlines_block:nnn
7604         { \exp_not:n { #5 } }
7605         { #1 - #2 }
7606         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7607     }
7608 }
7609 \bool_if:NT \l_@@_hlines_block_bool
7610 {
7611     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7612     {
7613         \@@_hlines_block:nnn
7614         { \exp_not:n { #5 } }
7615         { #1 - #2 }
7616         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7617     }
7618 }
7619 \bool_if:NF \l_@@_transparent_bool
7620 {
7621     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7622     {

```

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).

```

7623         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7624         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7625     }
7626 }
7627 \tl_if_empty:NF \l_@@_draw_tl
7628 {
7629     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7630     { \@@_error:n { hlines~with~color } }
7631     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7632     {
7633         \@@_stroke_block:nnn

```

#5 are the options

```

7634         { \exp_not:n { #5 } }
7635         { #1 - #2 }
7636         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7637     }
7638     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7639     { { #1 } { #2 } { #3 } { #4 } }
7640 }
7641 \clist_if_empty:NF \l_@@_borders_clist
7642 {
7643     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7644     {
7645         \@@_stroke_borders_block:nnn
7646         { \exp_not:n { #5 } }
7647         { #1 - #2 }
7648         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7649     }
7650 }
7651 \tl_if_empty:NF \l_@@_fill_tl
7652 {
7653     \@@_add_opacity_to_fill:
7654     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7655     {
7656         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

```

7657         { #1 - #2 }
7658         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7659         { \dim_use:N \l_@@_rounded_corners_dim }
7660     }
7661 }
7662 \seq_if_empty:NF \l_@@_tikz_seq
7663 {
7664     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7665     {
7666         \@@_block_tikz:nnnnn
7667         { \seq_use:Nn \l_@@_tikz_seq { , } }
7668         { #1 }
7669         { #2 }
7670         { \int_use:N \l_@@_last_row_int }
7671         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7672     }
7673 }
7674 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7675 {
7676     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7677     {
7678         \@@_actually_diagbox:nnnnnn
7679         { #1 }
7680         { #2 }
7681         { \int_use:N \l_@@_last_row_int }
7682         { \int_use:N \l_@@_last_col_int }
7683         { \exp_not:n { ##1 } }
7684         { \exp_not:n { ##2 } }
7685     }
7686 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\\
                        &      & two      & \\\
three                  & four & five     & \\\
six                    & seven & eight    & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7687 \pgfpicture
7688 \pgfrememberpicturepositiononpagetrue
7689 \pgf@relevantforpicturesizefalse
7690 \@@_qpoint:n { row - #1 }
7691 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7692 \@@_qpoint:n { col - #2 }
7693 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7694 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }

```

```

7695 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7696 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7697 \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.

```

7698 \@@_pgf_rect_node:nnnnn
7699 { \@@_env: - #1 - #2 - block }
7700 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7701 \str_if_empty:NF \l_@@_block_name_str
7702 {
7703   \pgfnodealias
7704   { \@@_env: - \l_@@_block_name_str }
7705   { \@@_env: - #1 - #2 - block }
7706   \str_if_empty:NF \l_@@_name_str
7707   {
7708     \pgfnodealias
7709     { \l_@@_name_str - \l_@@_block_name_str }
7710     { \@@_env: - #1 - #2 - block }
7711   }
7712 }

```

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.

```

7713 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7714 {
7715   \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.

```

7716 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7717 {

```

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.

```

7718 \cs_if_exist:cT
7719 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7720 {
7721   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7722   {
7723     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7724     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7725   }
7726 }
7727 }

```

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.

```

7728 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7729 {
7730   \@@_qpoint:n { col - #2 }
7731   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7732 }
7733 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7734 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7735 {
7736   \cs_if_exist:cT
7737   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7738   {
7739     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7740     {

```

```

7741         \pgfpointanchor
7742         { \@@_env: - #1 - \int_use:N \l_@@_last_col_int }
7743         { east }
7744         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7745     }
7746 }
7747 }
7748 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7749 {
7750     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7751     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7752 }
7753 \@@_pgf_rect_node:nnnnn
7754 { \@@_env: - #1 - #2 - block - short }
7755     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7756 }

```

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.

```

7757 \bool_if:NT \l_@@_medium_nodes_bool
7758 {
7759     \@@_pgf_rect_node:nnn
7760     { \@@_env: - #1 - #2 - block - medium }
7761     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7762     {
7763         \pgfpointanchor
7764         { \@@_env:
7765             - \int_use:N \l_@@_last_row_int
7766             - \int_use:N \l_@@_last_col_int - medium
7767         }
7768         { south-east }
7769     }
7770 }
7771 \endpgfpicture

```

```

7772 \bool_if:NTF \l_@@_ampersand_bool
7773 {
7774     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7775     \int_zero_new:N \l_@@_split_int
7776     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7777     \pgfpicture
7778     \pgfrememberpicturepositiononpagetrue
7779     \pgf@relevantforpicturesizefalse
7780
7781     \@@_qpoint:n { row - #1 }
7782     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7783     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7784     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7785     \@@_qpoint:n { col - #2 }
7786     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7787     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7788     \dim_set:Nn \l_tmpb_dim
7789     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7790     \bool_lazy_or:nnT
7791     \l_@@_vlines_block_bool
7792     { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7793     {
7794         \int_step_inline:nn { \l_@@_split_int - 1 }
7795         {
7796             \pgfpathmoveto
7797             {
7798                 \pgfpoint
7799                 { \l_tmpa_dim + ##1 \l_tmpb_dim }

```

```

7800         \l_@@_tmpc_dim
7801     }
7802     \pgfpathlineto
7803     {
7804         \pgfpoint
7805         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7806         \l_@@_tmpd_dim
7807     }
7808     \CT@arc@
7809     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7810     \pgfsetrectcap
7811     \pgfusepathqstroke
7812 }
7813 }
7814 \@@_qpoint:n { row - #1 - base }
7815 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7816 \int_step_inline:nn \l_@@_split_int
7817 {
7818     \group_begin:
7819     \dim_set:Nn \col@sep
7820     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7821     \pgftransformshift
7822     {
7823         \pgfpoint
7824         {
7825             \l_tmpa_dim + ##1 \l_tmpb_dim -
7826             \str_case:on \l_@@_hpos_block_str
7827             {
7828                 l { \l_tmpb_dim + \col@sep }
7829                 c { 0.5 \l_tmpb_dim }
7830                 r { \col@sep }
7831             }
7832         }
7833         { \l_@@_tmpc_dim }
7834     }
7835     \pgfset { inner~sep = \c_zero_dim }
7836     \pgfnode
7837     { rectangle }
7838     {
7839         \str_case:on \l_@@_hpos_block_str
7840         {
7841             c { base }
7842             l { base~west }
7843             r { base~east }
7844         }
7845     }
7846     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7847     \group_end:
7848 }
7849 \endpgfpicture
7850 }

```

Now the case where there is no ampersand & in the content of the block.

```

7851 {
7852     \bool_if:NTF \l_@@_p_block_bool
7853     {

```

When the final user has used the key p, we have to compute the width.

```

7854     \pgfpicture
7855     \pgfrememberpicturepositiononpagetrue
7856     \pgf@relevantforpicturesizefalse
7857     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7858     {
7859         \@@_qpoint:n { col - #2 }

```

```

7860         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7861         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7862     }
7863     {
7864         \pgfpointanchor { \l_@@_env: - #1 - #2 - block - short } { west }
7865         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7866         \pgfpointanchor { \l_@@_env: - #1 - #2 - block - short } { east }
7867     }
7868     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7869 \endpgfpicture
7870 \hbox_set:Nn \l_@@_cell_box
7871 {
7872     \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7873     { \g_tmpb_dim }
7874     \str_case:on \l_@@_hpos_block_str
7875     { c \centering r \raggedleft l \raggedright j { } }
7876     #6
7877     \end { minipage }
7878 }
7879 }
7880 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7881 \bool_if:NT \g_@@_rotate_bool \l_@@_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.

```

7882 \pgfpicture
7883 \pgfrememberpicturepositiononpagetrue
7884 \pgf@relevantforpicturesizefalse
7885 \bool_lazy_any:nTF
7886 {
7887     { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7888     { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7889     { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7890     { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7891 }
7892 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7893     \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`.

```

7894     \bool_if:nT \g_@@_last_col_found_bool
7895     {
7896         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7897         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7898     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7899     \tl_set:Ne \l_tmpa_tl
7900     {
7901         \str_case:on \l_@@_vpos_block_str
7902         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7903         { } { % added 2024-06-29
7904             \str_case:on \l_@@_hpos_block_str
7905             {
7906                 c { center }
7907                 l { west }
7908                 r { east }
7909                 j { center }
7910             }

```

```

7911     }
7912     c {
7913         \str_case:on \l_@@_hpos_block_str
7914         {
7915             c { center }
7916             l { west }
7917             r { east }
7918             j { center }
7919         }
7920
7921     }
7922     T {
7923         \str_case:on \l_@@_hpos_block_str
7924         {
7925             c { north }
7926             l { north~west }
7927             r { north~east }
7928             j { north }
7929         }
7930
7931     }
7932     B {
7933         \str_case:on \l_@@_hpos_block_str
7934         {
7935             c { south }
7936             l { south~west }
7937             r { south~east }
7938             j { south }
7939         }
7940
7941     }
7942 }
7943
7944 \pgftransformshift
7945 {
7946     \pgfpointanchor
7947     {
7948         \@@_env: - #1 - #2 - block
7949         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7950     }
7951     { \l_tmpa_tl }
7952 }
7953 \pgfset { inner~sep = \c_zero_dim }
7954 \pgfnode
7955 { rectangle }
7956 { \l_tmpa_tl }
7957 { \box_use_drop:N \l_@@_cell_box } { } { }
7958 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7959 {
7960     \pgfextracty \l_tmpa_dim
7961     {
7962         \@@_qpoint:n
7963         {
7964             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7965             - base
7966         }
7967     }
7968     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7969     \pgfpointanchor

```



```

7970         {
7971         \@@_env: - #1 - #2 - block
7972         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7973         }
7974         {
7975         \str_case:on \l_@@_hpos_block_str
7976         {
7977             c { center }
7978             l { west }
7979             r { east }
7980             j { center }
7981         }
7982     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

7983         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7984         \pgfset { inner~sep = \c_zero_dim }
7985         \pgfnode
7986         { rectangle }
7987         {
7988             \str_case:on \l_@@_hpos_block_str
7989             {
7990                 c { base }
7991                 l { base~west }
7992                 r { base~east }
7993                 j { base }
7994             }
7995         }
7996         { \box_use_drop:N \l_@@_cell_box } { } { }
7997     }
7998     \endpgfpicture
7999 }
8000 \group_end:
8001 }

```

For the command \cellcolor used within a sub-cell of a \Block (when the character & is used inside the cell).

```

8002 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8003 {
8004     \pgfpicture
8005     \pgfrememberpicturepositiononpagetrue
8006     \pgf@relevantforpicturesizefalse
8007     \pgfpathrectanglecorners
8008     { \pgfpoint { #2 } { #3 } }
8009     { \pgfpoint { #4 } { #5 } }
8010     \pgfsetfillcolor { #1 }
8011     \pgfusepath { fill }
8012     \endpgfpicture
8013 }

```

The following command adds the value of \l_@@_opacity_tl (if not empty) to the specification of color set in \l_@@_fill_tl (the information of opacity is added in between square brackets before the color itself).

```

8014 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8015 {
8016     \tl_if_empty:NF \l_@@_opacity_tl
8017     {
8018         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8019         {
8020             \tl_set:Nx \l_@@_fill_tl
8021             {
8022                 [ opacity = \l_@@_opacity_tl ,

```

```

8023         \tl_tail:o \l_@@_fill_tl
8024     }
8025 }
8026 {
8027     \tl_set:Ne \l_@@_fill_tl
8028     { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8029 }
8030 }
8031 }

```

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).

```

8032 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8033 {
8034     \group_begin:
8035     \tl_clear:N \l_@@_draw_tl
8036     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8037     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8038     \pgfpicture
8039     \pgfrememberpicturepositiononpagetrue
8040     \pgf@relevantforpicturesizefalse
8041     \tl_if_empty:NF \l_@@_draw_tl
8042     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8043         \tl_if_eq:NnTF \l_@@_draw_tl { default }
8044         { \CT@arc@ }
8045         { \@@_color:o \l_@@_draw_tl }
8046     }
8047     \pgfsetcornersarced
8048     {
8049         \pgfpoint
8050         { \l_@@_rounded_corners_dim }
8051         { \l_@@_rounded_corners_dim }
8052     }
8053     \@@_cut_on_hyphen:w #2 \q_stop
8054     \int_compare:nNnF \l_tmpa_tl > \c@iRow
8055     {
8056         \int_compare:nNnF \l_tmpb_tl > \c@jCol
8057         {
8058             \@@_qpoint:n { row - \l_tmpa_tl }
8059             \dim_set_eq:NN \l_tmpb_dim \pgf@y
8060             \@@_qpoint:n { col - \l_tmpb_tl }
8061             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8062             \@@_cut_on_hyphen:w #3 \q_stop
8063             \int_compare:nNnT \l_tmpa_tl > \c@iRow
8064             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8065             \int_compare:nNnT \l_tmpb_tl > \c@jCol
8066             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8067             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8068             \dim_set_eq:NN \l_tmpa_dim \pgf@y
8069             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8070             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8071             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8072             \pgfpathrectanglecorners
8073             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8074             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8075             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8076             { \pgfusepathqstroke }
8077             { \pgfusepath { stroke } }
8078         }

```

```

8079     }
8080     \endpgfpicture
8081     \group_end:
8082 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8083 \keys_define:nn { nicematrix / BlockStroke }
8084 {
8085     color .tl_set:N = \l_@@_draw_tl ,
8086     draw .code:n =
8087         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8088     draw .default:n = default ,
8089     line-width .dim_set:N = \l_@@_line_width_dim ,
8090     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8091     rounded-corners .default:n = 4 pt
8092 }

```

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).

```

8093 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8094 {
8095     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8096     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8097     \@@_cut_on_hyphen:w #2 \q_stop
8098     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8099     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8100     \@@_cut_on_hyphen:w #3 \q_stop
8101     \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8102     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8103     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8104     {
8105         \use:e
8106         {
8107             \@@_vline:n
8108             {
8109                 position = ##1 ,
8110                 start = \l_@@_tmpc_tl ,
8111                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
8112                 total-width = \dim_use:N \l_@@_line_width_dim
8113             }
8114         }
8115     }
8116 }
8117 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8118 {
8119     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8120     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8121     \@@_cut_on_hyphen:w #2 \q_stop
8122     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8123     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8124     \@@_cut_on_hyphen:w #3 \q_stop
8125     \tl_set:Nc \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8126     \tl_set:Nc \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8127     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8128     {
8129         \use:e
8130         {
8131             \@@_hline:n
8132             {
8133                 position = ##1 ,
8134                 start = \l_@@_tmpd_tl ,
8135                 end = \int_eval:n { \l_tmpb_tl - 1 } ,

```

```

8136         total-width = \dim_use:N \l_@@_line_width_dim
8137     }
8138 }
8139 }
8140 }

```

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).

```

8141 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8142 {
8143     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8144     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8145     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8146     { \@@_error:n { borders~forbidden } }
8147     {
8148         \tl_clear_new:N \l_@@_borders_tikz_tl
8149         \keys_set:no
8150         { nicematrix / OnlyForTikzInBorders }
8151         \l_@@_borders_clist
8152         \@@_cut_on_hyphen:w #2 \q_stop
8153         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8154         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8155         \@@_cut_on_hyphen:w #3 \q_stop
8156         \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8157         \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8158         \@@_stroke_borders_block_i:
8159     }
8160 }
8161 \hook_gput_code:nnn { begindocument } { . }
8162 {
8163     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8164     {
8165         \c_@@_pgfortikzpicture_tl
8166         \@@_stroke_borders_block_ii:
8167         \c_@@_endpgfortikzpicture_tl
8168     }
8169 }
8170 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8171 {
8172     \pgfrememberpicturepositiononpagetrue
8173     \pgf@relevantforpicturesizefalse
8174     \CT@arc@
8175     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8176     \clist_if_in:NnT \l_@@_borders_clist { right }
8177     { \@@_stroke_vertical:n \l_tmpb_tl }
8178     \clist_if_in:NnT \l_@@_borders_clist { left }
8179     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8180     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8181     { \@@_stroke_horizontal:n \l_tmpa_tl }
8182     \clist_if_in:NnT \l_@@_borders_clist { top }
8183     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8184 }
8185 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8186 {
8187     tikz .code:n =
8188         \cs_if_exist:NTF \tikzpicture
8189         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8190         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8191     tikz .value_required:n = true ,
8192     top .code:n = ,
8193     bottom .code:n = ,

```

```

8194     left .code:n = ,
8195     right .code:n = ,
8196     unknown .code:n = \@@_error:n { bad~border }
8197 }

```

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).

```

8198 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8199 {
8200   \@@_qpoint:n \l_@@_tmpc_tl
8201   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8202   \@@_qpoint:n \l_tmpa_tl
8203   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8204   \@@_qpoint:n { #1 }
8205   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8206   {
8207     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8208     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8209     \pgfusepathqstroke
8210   }
8211   {
8212     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8213       ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8214   }
8215 }

```

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).

```

8216 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8217 {
8218   \@@_qpoint:n \l_@@_tmpd_tl
8219   \clist_if_in:NnTF \l_@@_borders_clist { left }
8220   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8221   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8222   \@@_qpoint:n \l_tmpb_tl
8223   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8224   \@@_qpoint:n { #1 }
8225   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8226   {
8227     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8228     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8229     \pgfusepathqstroke
8230   }
8231   {
8232     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8233       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8234   }
8235 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8236 \keys_define:nn { nicematrix / BlockBorders }
8237 {
8238   borders .clist_set:N = \l_@@_borders_clist ,
8239   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8240   rounded-corners .default:n = 4 pt ,
8241   line-width .dim_set:N = \l_@@_line_width_dim
8242 }

```

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.

```

8243 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8244 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8245 {
8246   \begin { tikzpicture }
8247   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```

8248   \clist_map_inline:nn { #1 }
8249   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8250     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8251     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8252     (
8253     [
8254       xshift = \dim_use:N \l_@@_offset_dim ,
8255       yshift = - \dim_use:N \l_@@_offset_dim
8256     ]
8257     #2 -| #3
8258     )
8259     rectangle
8260     (
8261     [
8262       xshift = - \dim_use:N \l_@@_offset_dim ,
8263       yshift = \dim_use:N \l_@@_offset_dim
8264     ]
8265     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8266     ) ;
8267   }
8268   \end { tikzpicture }
8269 }

8270 \keys_define:nn { nicematrix / SpecialOffset }
8271 { 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.

```

8272 \cs_new_protected:Npn \@@_NullBlock:
8273 { \@@_collect_options:n { \@@_NullBlock_i: } }
8274 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8275 { }

```

27 How to draw the dotted lines transparently

```

8276 \cs_set_protected:Npn \@@_renew_matrix:
8277 {
8278   \RenewDocumentEnvironment { pmatrix } { } {
8279     { \pNiceMatrix }
8280     { \endpNiceMatrix }
8281   \RenewDocumentEnvironment { vmatrix } { } {
8282     { \vNiceMatrix }
8283     { \endvNiceMatrix }
8284   \RenewDocumentEnvironment { Vmatrix } { } {
8285     { \VNiceMatrix }
8286     { \endVNiceMatrix }

```

```

8287 \RenewDocumentEnvironment { bmatrix } { }
8288 { \bNiceMatrix }
8289 { \endbNiceMatrix }
8290 \RenewDocumentEnvironment { Bmatrix } { }
8291 { \BNiceMatrix }
8292 { \endBNiceMatrix }
8293 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8294 \keys_define:nn { nicematrix / Auto }
8295 {
8296   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8297   columns-type .value_required:n = true ,
8298   l .meta:n = { columns-type = l } ,
8299   r .meta:n = { columns-type = r } ,
8300   c .meta:n = { columns-type = c } ,
8301   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8302   delimiters / color .value_required:n = true ,
8303   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8304   delimiters / max-width .default:n = true ,
8305   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8306   delimiters .value_required:n = true ,
8307   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8308   rounded-corners .default:n = 4 pt
8309 }
8310 \NewDocumentCommand \AutoNiceMatrixWithDelims
8311 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8312 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8313 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8314 {

```

The group is for the protection of the keys.

```

8315 \group_begin:
8316 \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8317 \use:e
8318 {
8319   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8320   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8321   [ \exp_not:o \l_tmpa_tl ]
8322 }
8323 \int_if_zero:nT \l_@@_first_row_int
8324 {
8325   \int_if_zero:nT \l_@@_first_col_int { & }
8326   \prg_replicate:nn { #4 - 1 } { & }
8327   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8328 }
8329 \prg_replicate:nn { #3 }
8330 {
8331   \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`).

```

8332 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8333 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8334 }
8335 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8336 {
8337   \int_if_zero:nT \l_@@_first_col_int { & }

```

```

8338     \prg_replicate:nn { #4 - 1 } { & }
8339     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8340   }
8341   \end { NiceArrayWithDelims }
8342   \group_end:
8343 }
8344 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8345 {
8346   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8347   {
8348     \bool_gset_true:N \g_@@_delims_bool
8349     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8350     \AutoNiceMatrixWithDelims { #2 } { #3 }
8351   }
8352 }
8353 \@@_define_com:nnn p ( )
8354 \@@_define_com:nnn b [ ]
8355 \@@_define_com:nnn v | |
8356 \@@_define_com:nnn V \ | \ |
8357 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8358 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8359 {
8360   \group_begin:
8361   \bool_gset_false:N \g_@@_delims_bool
8362   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8363   \group_end:
8364 }

```

29 The redefinition of the command `\dotfill`

```

8365 \cs_set_eq:NN \@@_old_dotfill \dotfill
8366 \cs_new_protected:Npn \@@_dotfill:
8367 {

```

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}`).

```

8368   \@@_old_dotfill
8369   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8370 }

```

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 its 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`.

```

8371 \cs_new_protected:Npn \@@_dotfill_i:
8372 { \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.

```

8373 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8374 {
8375   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8376   {

```



```

8377 \@@_actually_diagbox:nnnnnn
8378 { \int_use:N \c@iRow }
8379 { \int_use:N \c@jCol }
8380 { \int_use:N \c@iRow }
8381 { \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.

```

8382 { \g_@@_row_style_tl \exp_not:n { #1 } }
8383 { \g_@@_row_style_tl \exp_not:n { #2 } }
8384 }

```

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`.

```

8385 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8386 {
8387   { \int_use:N \c@iRow }
8388   { \int_use:N \c@jCol }
8389   { \int_use:N \c@iRow }
8390   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8391 { }
8392 }
8393 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it's possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

8394 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8395 {
8396   \pgfpicture
8397   \pgf@relevantforpicturesizefalse
8398   \pgfrememberpicturepositiononpagetrue
8399   \@@_qpoint:n { row - #1 }
8400   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8401   \@@_qpoint:n { col - #2 }
8402   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8403   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8404   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8405   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8406   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8407   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8408   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8409   {

```

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.

```

8410 \CT@arc@
8411 \pgfsetroundcap
8412 \pgfusepathqstroke
8413 }
8414 \pgfset { inner~sep = 1 pt }
8415 \pgfscope
8416 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8417 \pgfnode { rectangle } { south~west }
8418 {
8419   \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8420     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8421     \end { minipage }
8422   }
8423   { }
8424   { }
8425   \endpgfscope
8426   \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8427   \pgfnode { rectangle } { north-east }
8428   {
8429     \begin { minipage } { 20 cm }
8430     \raggedleft
8431     \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8432     \end { minipage }
8433   }
8434   { }
8435   { }
8436   \endpgfpicture
8437 }

```

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`.

```

8438 \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 `\`.

```

8439 \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`.

```

8440 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8441 {
8442   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8443   \@@_CodeAfter_iv:n
8444 }

```

We catch the argument of the command `\end` (in #1).

```

8445 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8446 {

```

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.

```

8447   \str_if_eq:eeTF \currenvir { #1 }
8448   { \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`.

```

8449   {
8450     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8451     \@@_CodeAfter_ii:n
8452   }
8453 }

```

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.

```

8454 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8455 {
8456   \pgfpicture
8457   \pgfrememberpicturepositiononpagetrue
8458   \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.

```

8459   \@@_qpoint:n { row - 1 }
8460   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8461   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8462   \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).

```

8463   \bool_if:nTF { #3 }
8464   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8465   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8466   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8467   {
8468     \cs_if_exist:cT
8469     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8470     {
8471       \pgfpointanchor
8472       { \@@_env: - ##1 - #2 }
8473       { \bool_if:nTF { #3 } { west } { east } }
8474       \dim_set:Nn \l_tmpa_dim
8475       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8476     }
8477   }

```

Now we can put the delimiter with a node of PGF.

```

8478   \pgfset { inner~sep = \c_zero_dim }
8479   \dim_zero:N \nulldelimiterspace
8480   \pgftransformshift
8481   {
8482     \pgfpoint
8483     { \l_tmpa_dim }
8484     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8485   }
8486   \pgfnode
8487   { rectangle }
8488   { \bool_if:nTF { #3 } { east } { west } }
8489   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8490   \nullfont
8491   \c_math_toggle_token
8492   \@@_color:o \l_@@_delimiters_color_tl
8493   \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

8494     \vcenter
8495     {
8496         \nullfont
8497         \hrule \@height
8498             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8499             \@depth \c_zero_dim
8500             \@width \c_zero_dim
8501     }
8502     \bool_if:nTF { #3 } { \right . } { \right #1 }
8503     \c_math_toggle_token
8504 }
8505 { }
8506 { }
8507 \endpgfpicture
8508 }

```

33 The command \SubMatrix

```

8509 \keys_define:nn { nicematrix / sub-matrix }
8510 {
8511     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8512     extra-height .value_required:n = true ,
8513     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8514     left-xshift .value_required:n = true ,
8515     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8516     right-xshift .value_required:n = true ,
8517     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8518     xshift .value_required:n = true ,
8519     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8520     delimiters / color .value_required:n = true ,
8521     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8522     slim .default:n = true ,
8523     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8524     hlines .default:n = all ,
8525     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8526     vlines .default:n = all ,
8527     hvlines .meta:n = { hlines, vlines } ,
8528     hvlines .value_forbidden:n = true
8529 }
8530 \keys_define:nn { nicematrix }
8531 {
8532     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8533     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8534     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8535     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8536 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8537 \keys_define:nn { nicematrix / SubMatrix }
8538 {
8539     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8540     delimiters / color .value_required:n = true ,
8541     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8542     hlines .default:n = all ,
8543     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8544     vlines .default:n = all ,
8545     hvlines .meta:n = { hlines, vlines } ,
8546     hvlines .value_forbidden:n = true ,
8547     name .code:n =

```

```

8548 \tl_if_empty:nTF { #1 }
8549 { \@@_error:n { Invalid-name } }
8550 {
8551   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8552   {
8553     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8554     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8555     {
8556       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8557       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8558     }
8559   }
8560   { \@@_error:n { Invalid-name } }
8561 } ,
8562 name .value_required:n = true ,
8563 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8564 rules .value_required:n = true ,
8565 code .tl_set:N = \l_@@_code_tl ,
8566 code .value_required:n = true ,
8567 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8568 }

8569 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8570 {
8571   \peek_remove_spaces:n
8572   {
8573     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8574     {
8575       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8576       [
8577         delimiters / color = \l_@@_delimiters_color_tl ,
8578         hlines = \l_@@_submatrix_hlines_clist ,
8579         vlines = \l_@@_submatrix_vlines_clist ,
8580         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8581         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8582         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8583         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8584         #5
8585       ]
8586     }
8587     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8588   }
8589 }

8590 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8591 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8592 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8593 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8594 {
8595   \seq_gput_right:Ne \g_@@_submatrix_seq
8596   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8597   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8598   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8599   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8600   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8601 }
8602 }

```

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`.

```

8603 \hook_gput_code:nnn { begindocument } { . }
8604 {
8605   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8606   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8607   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8608     {
8609       \peek_remove_spaces:n
8610       {
8611         \@@_sub_matrix:nnnnnnn
8612         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8613       }
8614     }
8615 }

```

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).

```

8616 \NewDocumentCommand \@@_compute_i_j:nn
8617 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8618 { \@@_compute_i_j:nnnn #1 #2 }
8619 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8620 {
8621   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8622   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8623   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8624   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8625   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8626     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8627   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8628     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8629   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8630     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8631   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8632     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8633 }
8634 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8635 {
8636   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8637 \@@_compute_i_j:nn { #2 } { #3 }
8638 \int_compare:NnNT \l_@@_first_i_tl = \l_@@_last_i_tl
8639   { \cs_set_nopar:Npn \arraystretch { 1 } }
8640 \bool_lazy_or:nnTF
8641   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8642   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8643   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8644   {
8645     \str_clear_new:N \l_@@_submatrix_name_str
8646     \keys_set:nn { nicematrix / SubMatrix } { #5 }

```

```

8647 \pgfpicture
8648 \pgfrememberpicturepositiononpagetrue
8649 \pgf@relevantforpicturesizefalse
8650 \pgfset { inner~sep = \c_zero_dim }
8651 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8652 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8653 \bool_if:NTF \l_@@_submatrix_slim_bool
8654 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8655 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8656 {
8657   \cs_if_exist:cT
8658   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8659   {
8660     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8661     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8662     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8663   }
8664   \cs_if_exist:cT
8665   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8666   {
8667     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8668     \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8669     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8670   }
8671 }
8672 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8673 { \@@_error:nn { Impossible~delimiter } { left } }
8674 {
8675   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8676   { \@@_error:nn { Impossible~delimiter } { right } }
8677   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8678 }
8679 \endpgfpicture
8680 }
8681 \group_end:
8682 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8683 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8684 {
8685   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8686   \dim_set:Nn \l_@@_y_initial_dim
8687   {
8688     \fp_to_dim:n
8689     {
8690       \pgf@y
8691       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8692     }
8693   }
8694   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8695   \dim_set:Nn \l_@@_y_final_dim
8696   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8697   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8698   {
8699     \cs_if_exist:cT
8700     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8701     {
8702       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8703       \dim_set:Nn \l_@@_y_initial_dim
8704       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8705     }

```

```

8706 \cs_if_exist:cT
8707 { pgf @ sh @ ns @ \l_@@_env: - \l_@@_last_i_tl - ##1 }
8708 {
8709 \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
8710 \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8711 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8712 }
8713 }
8714 \dim_set:Nn \l_tmpa_dim
8715 {
8716 \l_@@_y_initial_dim - \l_@@_y_final_dim +
8717 \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8718 }
8719 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8720 \group_begin:
8721 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8722 \l_@@_set_CT@arc@:o \l_@@_rules_color_tl
8723 \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`.

```

8724 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8725 {
8726 \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8727 {
8728 \int_compare:nNnT
8729 { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8730 {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8731 \l_@@_qpoint:n { col - ##1 }
8732 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8733 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8734 \pgfusepathqstroke
8735 }
8736 }
8737 }

```

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.

```

8738 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8739 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8740 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8741 {
8742 \bool_lazy_and:nnTF
8743 { \int_compare_p:nNn { ##1 } > \c_zero_int }
8744 {
8745 \int_compare_p:nNn
8746 { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8747 {
8748 \l_@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8749 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8750 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8751 \pgfusepathqstroke
8752 }
8753 { \l_@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8754 }

```

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.


```

8755 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8756 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8757 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8758 {
8759   \bool_lazy_and:nnTF
8760   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8761   {
8762     \int_compare_p:nNn
8763     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8764   {
8765     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8766   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8767   \dim_set:Nn \l_tmpa_dim
8768   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8769   \str_case:nn { #1 }
8770   {
8771     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8772     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8773     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8774   }
8775   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8776   \dim_set:Nn \l_tmpb_dim
8777   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8778   \str_case:nn { #2 }
8779   {
8780     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8781     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8782     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8783   }
8784   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8785   \pgfusepathqstroke
8786   \group_end:
8787 }
8788 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8789 }

```

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).

```

8790 \str_if_empty:NF \l_@@_submatrix_name_str
8791 {
8792   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8793   \l_@@_x_initial_dim \l_@@_y_initial_dim
8794   \l_@@_x_final_dim \l_@@_y_final_dim
8795 }
8796 \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`.

```

8797 \begin { pgfscope }
8798 \pgftransformshift
8799 {
8800   \pgfpoint
8801   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8802   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8803 }
8804 \str_if_empty:NTF \l_@@_submatrix_name_str
8805 { \@@_node_left:nn #1 { } }

```

```

8806     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8807 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8808 \pgftransformshift
8809 {
8810   \pgfpoint
8811   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8812   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8813 }
8814 \str_if_empty:NTF \l_@@_submatrix_name_str
8815 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8816 {
8817   \@@_node_right:nnnn #2
8818   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8819 }
8820 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8821 \flag_clear_new:N \l_@@_code_flag
8822 \l_@@_code_tl
8823 }

```

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 $i-j$, $\text{row-}i$, $\text{col-}j$ 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`.

```

8824 \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.

```

8825 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8826 {
8827   \use:e
8828   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8829 }

```

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 -.

```

8830 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8831 { #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.

```

8832 \tl_const:Nn \c_@@_integers_alist_tl
8833 {
8834   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8835   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8836   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8837   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8838 }
8839 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8840 {

```

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`.

```

8841 \tl_if_empty:nTF { #2 }
8842 {
8843   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8844   {
8845     \flag_raise:N \l_@@_code_flag
8846     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8847     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8848     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8849   }
8850   { #1 }
8851 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8852 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8853 }

```

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`).

```

8854 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8855 {
8856   \str_case:nnF { #1 }
8857   {
8858     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8859     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8860   }

```

Now the case of a node of the form $i-j$.

```

8861 {
8862   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8863   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8864 }
8865 }

```

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`).

```

8866 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8867 {
8868   \pgfnode
8869   { rectangle }
8870   { east }
8871   {
8872     \nullfont
8873     \c_math_toggle_token
8874     \@@_color:o \l_@@_delimiters_color_tl
8875     \left #1
8876     \vcenter
8877     {
8878       \nullfont
8879       \hrule \@height \l_tmpa_dim
8880       \@depth \c_zero_dim
8881       \@width \c_zero_dim
8882     }
8883     \right .
8884     \c_math_toggle_token
8885   }
8886   { #2 }

```

```

8887 { }
8888 }

```

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.

```

8889 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8890 {
8891   \pgfnode
8892     { rectangle }
8893     { west }
8894     {
8895       \nullfont
8896       \c_math_toggle_token
8897       \colorlet { current-color } { . }
8898       \@@_color:o \l_@@_delimiters_color_tl
8899       \left .
8900       \vcenter
8901         {
8902           \nullfont
8903           \hrule \@height \l_tmpa_dim
8904             \@depth \c_zero_dim
8905             \@width \c_zero_dim
8906         }
8907       \right #1
8908       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8909       ~ { \color { current-color } \smash { #4 } }
8910       \c_math_toggle_token
8911     }
8912   { #2 }
8913   { }
8914 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8915 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8916 {
8917   \peek_remove_spaces:n
8918   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8919 }
8920 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8921 {
8922   \peek_remove_spaces:n
8923   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8924 }
8925 \keys_define:nn { nicematrix / Brace }
8926 {
8927   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8928   left-shorten .default:n = true ,
8929   left-shorten .value_forbidden:n = true ,
8930   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8931   right-shorten .default:n = true ,
8932   right-shorten .value_forbidden:n = true ,
8933   shorten .meta:n = { left-shorten , right-shorten } ,
8934   shorten .value_forbidden:n = true ,
8935   yshift .dim_set:N = \l_@@_brace_yshift_dim ,

```

```

8936 yshift .value_required:n = true ,
8937 yshift .initial:n = \c_zero_dim ,
8938 color .tl_set:N = \l_tmpa_tl ,
8939 color .value_required:n = true ,
8940 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8941 }

```

#1 is the first cell of the rectangle (with the syntax $i-j$; #2 is the last cell of the rectangle; #3 is the label of the text; #4 is the optional argument (a list of *key-value* pairs); #5 is equal to *under* or *over*.

```

8942 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8943 {
8944   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8945   \@@_compute_i_j:nn { #1 } { #2 }
8946   \bool_lazy_or:nnTF
8947   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8948   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8949   {
8950     \str_if_eq:eeTF { #5 } { under }
8951     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8952     { \@@_error:nn { Construct~too~large } { \OverBrace } }
8953   }
8954   {
8955     \tl_clear:N \l_tmpa_tl
8956     \keys_set:nn { nicematrix / Brace } { #4 }
8957     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8958     \pgfpicture
8959     \pgfrememberpicturepositiononpagetrue
8960     \pgf@relevantforpicturesizefalse
8961     \bool_if:NT \l_@@_brace_left_shorten_bool
8962     {
8963       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8964       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8965       {
8966         \cs_if_exist:cT
8967         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8968         {
8969           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8970
8971           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8972           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8973         }
8974       }
8975     }
8976     \bool_lazy_or:nnT
8977     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8978     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8979     {
8980       \@@_qpoint:n { col - \l_@@_first_j_tl }
8981       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8982     }
8983     \bool_if:NT \l_@@_brace_right_shorten_bool
8984     {
8985       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8986       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8987       {
8988         \cs_if_exist:cT
8989         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8990         {
8991           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8992           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8993           { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }

```

```

8994     }
8995   }
8996 }
8997 \bool_lazy_or:nnT
8998 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8999 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9000 {
9001   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9002   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9003 }
9004 \pgfset { inner~sep = \c_zero_dim }
9005 \str_if_eq:eeTF { #5 } { under }
9006 { \@@_underbrace_i:n { #3 } }
9007 { \@@_overbrace_i:n { #3 } }
9008 \endpgfpicture
9009 }
9010 \group_end:
9011 }

```

The argument is the text to put above the brace.

```

9012 \cs_new_protected:Npn \@@_overbrace_i:n #1
9013 {
9014   \@@_qpoint:n { row - \l_@@_first_i_tl }
9015   \pgftransformshift
9016   {
9017     \pgfpoint
9018     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9019     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9020   }
9021   \pgfnode
9022   { rectangle }
9023   { south }
9024   {
9025     \vtop
9026     {
9027       \group_begin:
9028       \everycr { }
9029       \halign
9030       {
9031         \hfil ## \hfil \crcr
9032         \bool_if:NTF \l_@@_tabular_bool
9033         { \begin { tabular } { c } #1 \end { tabular } }
9034         { $ \begin { array } { c } #1 \end { array } $ }
9035         \cr
9036         \c_math_toggle_token
9037         \overbrace
9038         {
9039           \hbox_to_wd:nn
9040           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9041           { }
9042         }
9043         \c_math_toggle_token
9044         \cr
9045       }
9046       \group_end:
9047     }
9048   }
9049   { }
9050   { }
9051 }

```

The argument is the text to put under the brace.

```

9052 \cs_new_protected:Npn \@@_underbrace_i:n #1

```

```

9053 {
9054   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9055   \pgftransformshift
9056   {
9057     \pgfpoint
9058     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9059     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9060   }
9061   \pgfnode
9062   { rectangle }
9063   { north }
9064   {
9065     \group_begin:
9066     \everycr { }
9067     \vbox
9068     {
9069       \halign
9070       {
9071         \hfil ## \hfil \crcr
9072         \c_math_toggle_token
9073         \underbrace
9074         {
9075           \hbox_to_wd:nn
9076           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9077           { }
9078         }
9079         \c_math_toggle_token
9080         \cr
9081         \bool_if:NTF \l_@@_tabular_bool
9082         { \begin { tabular } { c } #1 \end { tabular } }
9083         { $ \begin { array } { c } #1 \end { array } $ }
9084         \cr
9085       }
9086     }
9087     \group_end:
9088   }
9089   { }
9090   { }
9091 }

```

35 The command TikzEveryCell

```

9092 \bool_new:N \l_@@_not_empty_bool
9093 \bool_new:N \l_@@_empty_bool
9094
9095 \keys_define:nn { nicematrix / TikzEveryCell }
9096 {
9097   not-empty .code:n =
9098     \bool_lazy_or:nnTF
9099     \l_@@_in_code_after_bool
9100     \g_@@_recreate_cell_nodes_bool
9101     { \bool_set_true:N \l_@@_not_empty_bool }
9102     { \@@_error:n { detection~of~empty~cells } } ,
9103   not-empty .value_forbidden:n = true ,
9104   empty .code:n =
9105     \bool_lazy_or:nnTF
9106     \l_@@_in_code_after_bool
9107     \g_@@_recreate_cell_nodes_bool
9108     { \bool_set_true:N \l_@@_empty_bool }

```

```

9109     { \@@_error:n { detection~of~empty~cells } } ,
9110     empty .value_forbidden:n = true ,
9111     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9112 }
9113
9114
9115 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9116 {
9117     \IfPackageLoadedTF { tikz }
9118     {
9119         \group_begin:
9120         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9121     \tl_set:Nn \l_tmpa_tl { { #2 } }
9122     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9123     { \@@_for_a_block:nnnnn ##1 }
9124     \@@_all_the_cells:
9125     \group_end:
9126 }
9127 { \@@_error:n { TikzEveryCell~without~tikz } }
9128 }
9129
9130 \tl_new:N \@@_i_tl
9131 \tl_new:N \@@_j_tl
9132
9133
9134 \cs_new_protected:Nn \@@_all_the_cells:
9135 {
9136     \int_step_variable:nNn \c@iRow \@@_i_tl
9137     {
9138         \int_step_variable:nNn \c@jCol \@@_j_tl
9139         {
9140             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9141             {
9142                 \clist_if_in:NcF \l_@@_corners_cells_clist
9143                 { \@@_i_tl - \@@_j_tl }
9144                 {
9145                     \bool_set_false:N \l_tmpa_bool
9146                     \cs_if_exist:cTF
9147                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9148                     {
9149                         \bool_if:NF \l_@@_empty_bool
9150                         { \bool_set_true:N \l_tmpa_bool }
9151                     }
9152                     {
9153                         \bool_if:NF \l_@@_not_empty_bool
9154                         { \bool_set_true:N \l_tmpa_bool }
9155                     }
9156                     \bool_if:NT \l_tmpa_bool
9157                     {
9158                         \@@_block_tikz:nnnnn
9159                         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9160                     }
9161                 }
9162             }
9163         }
9164     }
9165 }
9166
9167 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9168 {
9169     \bool_if:NF \l_@@_empty_bool

```



```

9170     {
9171         \@@_block_tikz:onnnn
9172         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9173     }
9174     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9175 }
9176
9177 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9178 {
9179     \int_step_inline:nnn { #1 } { #3 }
9180     {
9181         \int_step_inline:nnn { #2 } { #4 }
9182         { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9183     }
9184 }

```

36 The command \ShowCellNames

```

9185 \NewDocumentCommand \@@_ShowCellNames { }
9186 {
9187     \bool_if:NT \l_@@_in_code_after_bool
9188     {
9189         \pgfpicture
9190         \pgfrememberpicturepositiononpagetrue
9191         \pgf@relevantforpicturesizefalse
9192         \pgfpathrectanglecorners
9193         { \@@_qpoint:n { 1 } }
9194         {
9195             \@@_qpoint:n
9196             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9197         }
9198         \pgfsetfillopacity { 0.75 }
9199         \pgfsetfillcolor { white }
9200         \pgfusepathqfill
9201         \endpgfpicture
9202     }
9203     \dim_gzero_new:N \g_@@_tmpc_dim
9204     \dim_gzero_new:N \g_@@_tmpd_dim
9205     \dim_gzero_new:N \g_@@_tmpe_dim
9206     \int_step_inline:nn \c@iRow
9207     {
9208         \bool_if:NTF \l_@@_in_code_after_bool
9209         {
9210             \pgfpicture
9211             \pgfrememberpicturepositiononpagetrue
9212             \pgf@relevantforpicturesizefalse
9213         }
9214         { \begin { pgfpicture } }
9215         \@@_qpoint:n { row - ##1 }
9216         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9217         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9218         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9219         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9220         \bool_if:NTF \l_@@_in_code_after_bool
9221         { \endpgfpicture }
9222         { \end { pgfpicture } }
9223         \int_step_inline:nn \c@jCol
9224         {
9225             \hbox_set:Nn \l_tmpa_box
9226             {
9227                 \normalfont \Large \sffamily \bfseries
9228                 \bool_if:NTF \l_@@_in_code_after_bool

```

```

9229         { \color { red } }
9230         { \color { red ! 50 } }
9231     ##1 - ####1
9232 }
9233 \bool_if:NTF \l_@@_in_code_after_bool
9234 {
9235     \pgfpicture
9236     \pgfrememberpicturepositiononpagetrue
9237     \pgf@relevantforpicturesizefalse
9238 }
9239 { \begin { pgfpicture } }
9240 @@_qpoint:n { col - ####1 }
9241 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9242 @@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9243 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9244 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9245 \bool_if:NTF \l_@@_in_code_after_bool
9246 { \endpgfpicture }
9247 { \end { pgfpicture } }
9248 \fp_set:Nn \l_tmpa_fp
9249 {
9250     \fp_min:nn
9251     {
9252         \fp_min:nn
9253         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9254         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9255     }
9256     { 1.0 }
9257 }
9258 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9259 \pgfpicture
9260 \pgfrememberpicturepositiononpagetrue
9261 \pgf@relevantforpicturesizefalse
9262 \pgftransformshift
9263 {
9264     \pgfpoint
9265     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9266     { \dim_use:N \g_tmpa_dim }
9267 }
9268 \pgfnode
9269 { rectangle }
9270 { center }
9271 { \box_use:N \l_tmpa_box }
9272 { }
9273 { }
9274 \endpgfpicture
9275 }
9276 }
9277 }

```

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.

```

9278 \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.

```

9279 \bool_new:N \g_@@_footnote_bool
9280 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9281 {
9282   The~key~'\l_keys_key_str'~is~unknown. \\
9283   That~key~will~be~ignored. \\
9284   For~a~list~of~the~available~keys,~type~H~<return>.
9285 }
9286 {
9287   The~available~keys~are~(in~alphabetic~order):~
9288   footnote,~
9289   footnotehyper,~
9290   messages~for~Overleaf,~
9291   renew~dots,~and~
9292   renew~matrix.
9293 }
9294 \keys_define:nn { nicematrix / Package }
9295 {
9296   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9297   renew~dots .value_forbidden:n = true ,
9298   renew~matrix .code:n = \@@_renew_matrix: ,
9299   renew~matrix .value_forbidden:n = true ,
9300   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9301   footnote .bool_set:N = \g_@@_footnote_bool ,
9302   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.

```

9303   no~test~for~array .code:n = \prg_do_nothing: ,
9304   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9305 }
9306 \ProcessKeysOptions { nicematrix / Package }

9307 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9308 {
9309   You~can't~use~the~option~'footnote'~because~the~package~
9310   footnotehyper~has~already~been~loaded.~
9311   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9312   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9313   of~the~package~footnotehyper.\\
9314   The~package~footnote~won't~be~loaded.
9315 }
9316 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9317 {
9318   You~can't~use~the~option~'footnotehyper'~because~the~package~
9319   footnote~has~already~been~loaded.~
9320   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9321   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9322   of~the~package~footnote.\\
9323   The~package~footnotehyper~won't~be~loaded.
9324 }

9325 \bool_if:NT \g_@@_footnote_bool
9326 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9327 \IfClassLoadedTF { beamer }
9328 { \bool_set_false:N \g_@@_footnote_bool }
9329 {

```

```

9330     \IfPackageLoadedTF { footnotehyper }
9331     { \@@_error:n { footnote~with~footnotehyper~package } }
9332     { \usepackage { footnote } }
9333   }
9334 }
9335 \bool_if:NT \g_@@_footnotehyper_bool
9336 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9337   \IfClassLoadedTF { beamer }
9338   { \bool_set_false:N \g_@@_footnote_bool }
9339   {
9340     \IfPackageLoadedTF { footnote }
9341     { \@@_error:n { footnotehyper~with~footnote~package } }
9342     { \usepackage { footnotehyper } }
9343   }
9344   \bool_set_true:N \g_@@_footnote_bool
9345 }

```

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.

```

9346 \bool_new:N \l_@@_underscore_loaded_bool
9347 \IfPackageLoadedT { underscore }
9348 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9349 \hook_gput_code:nnn { begindocument } { . }
9350 {
9351   \bool_if:NF \l_@@_underscore_loaded_bool
9352   {
9353     \IfPackageLoadedT { underscore }
9354     { \@@_error:n { underscore~after~nicematrix } }
9355   }
9356 }

```

39 Error messages of the package

```

9357 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9358 { \str_const:Nn \c_@@_available_keys_str { } }
9359 {
9360   \str_const:Nn \c_@@_available_keys_str
9361   { For~a~list~of~the~available~keys,~type~H~<return>. }
9362 }
9363 \seq_new:N \g_@@_types_of_matrix_seq
9364 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9365 {
9366   NiceMatrix ,
9367   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9368 }
9369 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq

```

```
9370 { \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`.

```
9371 \cs_new_protected:Npn \@@_error_too_much_cols:
9372 {
9373   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9374   { \@@_fatal:nn { too-much-cols-for-array } }
9375   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9376   { \@@_fatal:n { too-much-cols-for-matrix } }
9377   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9378   { \@@_fatal:n { too-much-cols-for-matrix } }
9379   \bool_if:NF \l_@@_last_col_without_value_bool
9380   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9381 }
```

The following command must *not* be protected since it's used in an error message.

```
9382 \cs_new:Npn \@@_message_hdotsfor:
9383 {
9384   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9385   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9386 }
9387 \@@_msg_new:nn { hvlines,~rounded-corners-and-corners }
9388 {
9389   Incompatible-options.\\
9390   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9391   The~output~will~not~be~reliable.
9392 }
9393 \@@_msg_new:nn { key~color~inside }
9394 {
9395   Key-deprecated.\\
9396   The~key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9397   and~have~been~deprecated.\\
9398   You~won't~have~similar~message~till~the~end~of~the~document.
9399 }
9400 \@@_msg_new:nn { negative~weight }
9401 {
9402   Negative-weight.\\
9403   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9404   the~value~'\int_use:N \l_@@_weight_int'.\\
9405   The~absolute~value~will~be~used.
9406 }
9407 \@@_msg_new:nn { last~col~not~used }
9408 {
9409   Column~not~used.\\
9410   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9411   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9412 }
9413 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9414 {
9415   Too-much-columns.\\
9416   In~the~row~\int_eval:n { \c@iRow },~
9417   you~try~to~use~more~columns~
9418   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\\
9419   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9420   (plus~the~exterior~columns).~This~error~is~fatal.
9421 }
9422 \@@_msg_new:nn { too-much-cols-for-matrix }
9423 {
```

```

9424 Too-much-columns.\\
9425 In-the-row~\int_eval:n { \c@iRow },~
9426 you-try-to-use-more-columns-than-allowed-by-your~
9427 \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
9428 number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
9429 columns)~is-fixed-by-the-LaTeX-counter~'MaxMatrixCols'.~
9430 Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9431 \token_to_str:N \setcounter\ to-change-that-value).~
9432 This-error-is-fatal.
9433 }

9434 \@@_msg_new:nn { too-much-cols-for-array }
9435 {
9436   Too-much-columns.\\
9437   In-the-row~\int_eval:n { \c@iRow },~
9438   ~you-try-to-use-more-columns-than-allowed-by-your~
9439   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9440   \int_use:N \g_@@_static_num_of_col_int
9441   \bool_if:nT
9442     { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9443     { ~(plus-the-exterior-ones) }.~
9444   This-error-is-fatal.
9445 }

9446 \@@_msg_new:nn { columns-not-used }
9447 {
9448   Columns-not-used.\\
9449   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9450   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9451   The-columns-you-did-not-used-won't-be-created.\\
9452   You-won't-have-similar-error-message-till-the-end-of-the-document.
9453 }

9454 \@@_msg_new:nn { empty-preamble }
9455 {
9456   Empty-preamble.\\
9457   The-preamble-of-your~\@@_full_name_env:\ is-empty.\\
9458   This-error-is-fatal.
9459 }

9460 \@@_msg_new:nn { in-first-col }
9461 {
9462   Erroneous-use.\\
9463   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9464   That-command-will-be-ignored.
9465 }

9466 \@@_msg_new:nn { in-last-col }
9467 {
9468   Erroneous-use.\\
9469   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9470   That-command-will-be-ignored.
9471 }

9472 \@@_msg_new:nn { in-first-row }
9473 {
9474   Erroneous-use.\\
9475   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9476   That-command-will-be-ignored.
9477 }

9478 \@@_msg_new:nn { in-last-row }
9479 {
9480   Erroneous-use.\\
9481   You-can't-use-the-command~#1 in-the-last-row~(exterior)~of-the-array.\\
9482   That-command-will-be-ignored.
9483 }

```

```

9484 \@@_msg_new:nn { TopRule~without~booktabs }
9485 {
9486   Erroneous~use.\\
9487   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9488   That~command~will~be~ignored.
9489 }
9490 \@@_msg_new:nn { TopRule~without~tikz }
9491 {
9492   Erroneous~use.\\
9493   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9494   That~command~will~be~ignored.
9495 }
9496 \@@_msg_new:nn { caption~outside~float }
9497 {
9498   Key~caption~forbidden.\\
9499   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9500   environment.~This~key~will~be~ignored.
9501 }
9502 \@@_msg_new:nn { short-caption~without~caption }
9503 {
9504   You~should~not~use~the~key~'short-caption'~without~'caption'.~
9505   However,~your~'short-caption'~will~be~used~as~'caption'.
9506 }
9507 \@@_msg_new:nn { double~closing~delimiter }
9508 {
9509   Double~delimiter.\\
9510   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9511   delimiter.~This~delimiter~will~be~ignored.
9512 }
9513 \@@_msg_new:nn { delimiter~after~opening }
9514 {
9515   Double~delimiter.\\
9516   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9517   delimiter.~That~delimiter~will~be~ignored.
9518 }
9519 \@@_msg_new:nn { bad~option~for~line~style }
9520 {
9521   Bad~line~style.\\
9522   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9523   is~'standard'.~That~key~will~be~ignored.
9524 }
9525 \@@_msg_new:nn { Identical~notes~in~caption }
9526 {
9527   Identical~tabular~notes.\\
9528   You~can't~put~several~notes~with~the~same~content~in~
9529   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9530   If~you~go~on,~the~output~will~probably~be~erroneous.
9531 }
9532 \@@_msg_new:nn { tabularnote~below~the~tabular }
9533 {
9534   \token_to_str:N \tabularnote\ forbidden\\
9535   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9536   of~your~tabular~because~the~caption~will~be~composed~below~
9537   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9538   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9539   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9540   no~similar~error~will~raised~in~this~document.
9541 }
9542 \@@_msg_new:nn { Unknown~key~for~rules }
9543 {

```

```

9544     Unknown~key.\\
9545     There~is~only~two~keys~available~here:~width~and~color.\\
9546     Your~key~'\l_keys_key_str'~will~be~ignored.
9547 }

9548 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9549 {
9550     Unknown~key.\\
9551     There~is~only~two~keys~available~here:~
9552     'empty'~and~'not-empty'.\\
9553     Your~key~'\l_keys_key_str'~will~be~ignored.
9554 }

9555 \@@_msg_new:nn { Unknown~key~for~rotate }
9556 {
9557     Unknown~key.\\
9558     The~only~key~available~here~is~'c'.\\
9559     Your~key~'\l_keys_key_str'~will~be~ignored.
9560 }

9561 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9562 {
9563     Unknown~key.\\
9564     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9565     It~you~go~on,~you~will~probably~have~other~errors. \\
9566     \c_@@_available_keys_str
9567 }
9568 {
9569     The~available~keys~are~(in~alphabetic~order):~
9570     ccommand,~
9571     color,~
9572     command,~
9573     dotted,~
9574     letter,~
9575     multiplicity,~
9576     sep~color,~
9577     tikz,~and~total~width.
9578 }

9579 \@@_msg_new:nnn { Unknown~key~for~xdots }
9580 {
9581     Unknown~key.\\
9582     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9583     \c_@@_available_keys_str
9584 }
9585 {
9586     The~available~keys~are~(in~alphabetic~order):~
9587     'color',~
9588     'horizontal~labels',~
9589     'inter',~
9590     'line~style',~
9591     'radius',~
9592     'shorten',~
9593     'shorten~end'~and~'shorten~start'.
9594 }

9595 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9596 {
9597     Unknown~key.\\
9598     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
9599     (and~you~try~to~use~'\l_keys_key_str')\\
9600     That~key~will~be~ignored.
9601 }

9602 \@@_msg_new:nn { label~without~caption }
9603 {
9604     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~

```



```

9605     you-have-not-used-the-key~'caption'~.~The-key~'label'~will-be-ignored.
9606 }
9607 \@@_msg_new:nn { W~warning }
9608 {
9609     Line~\msg_line_number:~.~The-cell-is-too-wide-for-your-column~'W'~
9610     (row~\int_use:N \c@iRow).
9611 }
9612 \@@_msg_new:nn { Construct-too-large }
9613 {
9614     Construct-too-large.\\
9615     Your-command~\token_to_str:N #1
9616     can't-be-drawn-because-your-matrix-is-too-small.\\
9617     That-command-will-be-ignored.
9618 }
9619 \@@_msg_new:nn { underscore-after-nicematrix }
9620 {
9621     Problem-with~'underscore'.\\
9622     The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9623     You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9624     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9625 }
9626 \@@_msg_new:nn { ampersand-in-light-syntax }
9627 {
9628     Ampersand~forbidden.\\
9629     You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9630     ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9631 }
9632 \@@_msg_new:nn { double-backslash-in-light-syntax }
9633 {
9634     Double-backslash~forbidden.\\
9635     You-can't-use~\token_to_str:N
9636     \\~to-separate-rows-because-the-key~'light-syntax'~
9637     is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9638     (set-by-the-key~'end-of-row').~This-error-is-fatal.
9639 }
9640 \@@_msg_new:nn { hlines-with-color }
9641 {
9642     Incompatible-keys.\\
9643     You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for-a~
9644     '\token_to_str:N \Block'~when-the-key~'color'~or~'draw'~is-used.\\
9645     However,~you-can-put-several-commands~\token_to_str:N \Block.\\
9646     Your-key-will-be-discarded.
9647 }
9648 \@@_msg_new:nn { bad-value-for-baseline }
9649 {
9650     Bad-value~for~baseline.\\
9651     The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
9652     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int\ and~
9653     \int_use:N \g_@@_row_total_int\ or-equal-to~'t',~'c'~or~'b'~or-of~
9654     the-form~'line-i'.\\
9655     A~value-of~1~will-be-used.
9656 }
9657 \@@_msg_new:nn { detection-of-empty-cells }
9658 {
9659     Problem-with~'not-empty'\\
9660     For-technical-reasons,~you-must-activate~
9661     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9662     in-order-to-use-the-key~'\l_keys_key_str'.\\
9663     That-key-will-be-ignored.
9664 }

```

```

9665 \@@_msg_new:nn { siunitx-not-loaded }
9666 {
9667   siunitx-not-loaded\\
9668   You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\\
9669   That-error-is-fatal.
9670 }
9671 \@@_msg_new:nn { Invalid-name }
9672 {
9673   Invalid-name.\\
9674   You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
9675   \SubMatrix\ of-your~\@@_full_name_env:.\\
9676   A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9677   This-key-will-be-ignored.
9678 }
9679 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9680 {
9681   Wrong-line.\\
9682   You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
9683   \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
9684   number-is-not-valid.~It-will-be-ignored.
9685 }
9686 \@@_msg_new:nn { Impossible-delimiter }
9687 {
9688   Impossible-delimiter.\\
9689   It's-impossible-to-draw-the~#1~delimiter-of-your~
9690   \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9691   in-that-column.
9692   \bool_if:NT \l_@@_submatrix_slim_bool
9693     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9694   This~\token_to_str:N \SubMatrix\ will-be-ignored.
9695 }
9696 \@@_msg_new:nnn { width-without-X-columns }
9697 {
9698   You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9699   That-key-will-be-ignored.
9700 }
9701 {
9702   This-message-is-the-message~'width-without-X-columns'~
9703   of-the-module~'nicematrix'.~
9704   The-experimented-users-can-disable-that-message-with~
9705   \token_to_str:N \msg_redirect_name:nnn.\\
9706 }
9707
9708 \@@_msg_new:nn { key-multiplicity-with-dotted }
9709 {
9710   Incompatible-keys. \\
9711   You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9712   in-a~'custom-line'.~They-are-incompatible. \\
9713   The-key~'multiplicity'~will-be-discarded.
9714 }
9715 \@@_msg_new:nn { empty-environment }
9716 {
9717   Empty-environment.\\
9718   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9719 }
9720 \@@_msg_new:nn { No-letter-and-no-command }
9721 {
9722   Erroneous-use.\\
9723   Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9724   key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9725   ~'ccommand'~(to-draw-horizontal-rules).\\

```

```

9726     However,~you~can~go~on.
9727 }
9728 \@@_msg_new:nn { Forbidden~letter }
9729 {
9730     Forbidden~letter.\\
9731     You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9732     It~will~be~ignored.
9733 }
9734 \@@_msg_new:nn { Several~letters }
9735 {
9736     Wrong~name.\\
9737     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9738     have~used~'\l_@@_letter_str').\\
9739     It~will~be~ignored.
9740 }
9741 \@@_msg_new:nn { Delimiter~with~small }
9742 {
9743     Delimiter~forbidden.\\
9744     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
9745     because~the~key~'small'~is~in~force.\\
9746     This~error~is~fatal.
9747 }
9748 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9749 {
9750     Unknown~cell.\\
9751     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9752     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
9753     can't~be~executed~because~a~cell~doesn't~exist.\\
9754     This~command~\token_to_str:N \line\ will~be~ignored.
9755 }
9756 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9757 {
9758     Duplicate~name.\\
9759     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
9760     in~this~\@@_full_name_env:.\\
9761     This~key~will~be~ignored.\\
9762     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9763     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9764 }
9765 {
9766     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9767     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9768 }
9769 \@@_msg_new:nn { r~or~l~with~preamble }
9770 {
9771     Erroneous~use.\\
9772     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9773     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9774     your~\@@_full_name_env:.\\
9775     This~key~will~be~ignored.
9776 }
9777 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9778 {
9779     Erroneous~use.\\
9780     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9781     the~array.~This~error~is~fatal.
9782 }
9783 \@@_msg_new:nn { bad~corner }
9784 {
9785     Bad~corner.\\

```

```

9786     #1-is-an-incorrect-specification-for-a-corner-(in-the-key~
9787     'corners').~The-available-values-are:~NW,~SW,~NE-and-SE.\\
9788     This-specification-of~corner~will~be~ignored.
9789 }
9790 \@@_msg_new:nn { bad~border }
9791 {
9792     Bad~border.\\
9793     \l_keys_key_str\space-is-an-incorrect-specification-for-a~border~
9794     (in-the-key~'borders'~of~the~command~\token_to_str:N \Block).~
9795     The-available-values-are:~left,~right,~top-and-bottom~(and-you-can~
9796     also-use-the-key~'tikz'
9797     \IfPackageLoadedF { tikz }
9798     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9799     This-specification-of~border~will~be~ignored.
9800 }
9801 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9802 {
9803     TikZ~not~loaded.\\
9804     You-can't-use~\token_to_str:N \TikzEveryCell\
9805     because-you-have-not~loaded~tikz.~
9806     This-command~will~be~ignored.
9807 }
9808 \@@_msg_new:nn { tikz-key~without~tikz }
9809 {
9810     TikZ~not~loaded.\\
9811     You-can't-use-the-key~'tikz'~for-the-command~'\token_to_str:N
9812     \Block'~because-you-have-not~loaded~tikz.~
9813     This-key~will~be~ignored.
9814 }
9815 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9816 {
9817     Erroneous-use.\\
9818     In-the~\@@_full_name_env:,~you~must~use~the~key~
9819     'last-col'~without~value.\\
9820     However,~you~can~go~on~for~this~time~
9821     (the-value~'\l_keys_value_tl'~will~be~ignored).
9822 }
9823 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9824 {
9825     Erroneous-use.\\
9826     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9827     'last-col'~without~value.\\
9828     However,~you~can~go~on~for~this~time~
9829     (the-value~'\l_keys_value_tl'~will~be~ignored).
9830 }
9831 \@@_msg_new:nn { Block~too~large~1 }
9832 {
9833     Block~too~large.\\
9834     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9835     too~small~for~that~block. \\
9836     This~block~and~maybe~others~will~be~ignored.
9837 }
9838 \@@_msg_new:nn { Block~too~large~2 }
9839 {
9840     Block~too~large.\\
9841     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9842     \g_@@_static_num_of_col_int\
9843     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9844     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9845     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:~\\
9846     This~block~and~maybe~others~will~be~ignored.

```

```

9847 }
9848 \@@_msg_new:nn { unknown~column~type }
9849 {
9850   Bad~column~type.\\
9851   The~column~type~'#1'~in~your~\@@_full_name_env:\
9852   is~unknown. \\
9853   This~error~is~fatal.
9854 }
9855 \@@_msg_new:nn { unknown~column~type~S }
9856 {
9857   Bad~column~type.\\
9858   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9859   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9860   load~that~package. \\
9861   This~error~is~fatal.
9862 }
9863 \@@_msg_new:nn { tabularnote~forbidden }
9864 {
9865   Forbidden~command.\\
9866   You~can't~use~the~command~\token_to_str:N\tabularnote\
9867   ~here.~This~command~is~available~only~in~
9868   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9869   the~argument~of~a~command~\token_to_str:N \caption\ included~
9870   in~an~environment~{table}. \\
9871   This~command~will~be~ignored.
9872 }
9873 \@@_msg_new:nn { borders~forbidden }
9874 {
9875   Forbidden~key.\\
9876   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9877   because~the~option~'rounded~corners'~
9878   is~in~force~with~a~non~zero~value.\\
9879   This~key~will~be~ignored.
9880 }
9881 \@@_msg_new:nn { bottomrule~without~booktabs }
9882 {
9883   booktabs~not~loaded.\\
9884   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9885   loaded~'booktabs'.\\
9886   This~key~will~be~ignored.
9887 }
9888 \@@_msg_new:nn { enumitem~not~loaded }
9889 {
9890   enumitem~not~loaded.\\
9891   You~can't~use~the~command~\token_to_str:N\tabularnote\
9892   ~because~you~haven't~loaded~'enumitem'.\\
9893   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9894   ignored~in~the~document.
9895 }
9896 \@@_msg_new:nn { tikz~without~tikz }
9897 {
9898   Tikz~not~loaded.\\
9899   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9900   loaded.~If~you~go~on,~that~key~will~be~ignored.
9901 }
9902 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9903 {
9904   Tikz~not~loaded.\\
9905   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9906   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~

```

```

9907     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9908     use~that~custom~line.
9909 }
9910 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9911 {
9912     Tikz~not~loaded.\\
9913     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9914     command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9915     That~key~will~be~ignored.
9916 }
9917 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9918 {
9919     Erroneous~use.\\
9920     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9921     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9922     The~key~'color'~will~be~discarded.
9923 }
9924 \@@_msg_new:nn { Wrong~last~row }
9925 {
9926     Wrong~number.\\
9927     You~have~used~'last~row'=\int_use:N \l_@@_last_row_int'~but~your~
9928     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9929     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9930     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9931     without~value~(more~compilations~might~be~necessary).
9932 }
9933 \@@_msg_new:nn { Yet~in~env }
9934 {
9935     Nested~environments.\\
9936     Environments~of~nicematrix~can't~be~nested.\\
9937     This~error~is~fatal.
9938 }
9939 \@@_msg_new:nn { Outside~math~mode }
9940 {
9941     Outside~math~mode.\\
9942     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9943     (and~not~in~\token_to_str:N \vcenter).\\
9944     This~error~is~fatal.
9945 }
9946 \@@_msg_new:nn { One~letter~allowed }
9947 {
9948     Bad~name.\\
9949     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9950     It~will~be~ignored.
9951 }
9952 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9953 {
9954     Environment~{TabularNote}~forbidden.\\
9955     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9956     but~*before*~the~\token_to_str:N \CodeAfter.\\
9957     This~environment~{TabularNote}~will~be~ignored.
9958 }
9959 \@@_msg_new:nn { varwidth~not~loaded }
9960 {
9961     varwidth~not~loaded.\\
9962     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9963     loaded.\\
9964     Your~column~will~behave~like~'p'.
9965 }
9966 \@@_msg_new:nnn { Unknow~key~for~RulesBis }

```

```

9967 {
9968   Unknown~key.\\
9969   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9970   \c_@@_available_keys_str
9971 }
9972 {
9973   The~available~keys~are~(in~alphabetic~order):~
9974   color,~
9975   dotted,~
9976   multiplicity,~
9977   sep-color,~
9978   tikz,~and~total-width.
9979 }
9980
9981 \@@_msg_new:nnn { Unknown~key~for~Block }
9982 {
9983   Unknown~key.\\
9984   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9985   \Block.\\ It~will~be~ignored. \\
9986   \c_@@_available_keys_str
9987 }
9988 {
9989   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
9990   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9991   opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9992   and~vlines.
9993 }
9994 \@@_msg_new:nnn { Unknown~key~for~Brace }
9995 {
9996   Unknown~key.\\
9997   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9998   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9999   It~will~be~ignored. \\
10000   \c_@@_available_keys_str
10001 }
10002 {
10003   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10004   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10005   right-shorten)~and~yshift.
10006 }
10007 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10008 {
10009   Unknown~key.\\
10010   The~key~'\l_keys_key_str'~is~unknown.\\
10011   It~will~be~ignored. \\
10012   \c_@@_available_keys_str
10013 }
10014 {
10015   The~available~keys~are~(in~alphabetic~order):~
10016   delimiters/color,~
10017   rules~(with~the~subkeys~'color'~and~'width'),~
10018   sub-matrix~(several~subkeys)~
10019   and~xdots~(several~subkeys).~
10020   The~latter~is~for~the~command~\token_to_str:N \line.
10021 }
10022 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10023 {
10024   Unknown~key.\\
10025   The~key~'\l_keys_key_str'~is~unknown.\\
10026   It~will~be~ignored. \\
10027   \c_@@_available_keys_str
10028 }

```

```

10029 {
10030     The~available~keys~are~(in~alphabetic~order):~
10031     create~cell~nodes,~
10032     delimiters/color~and~
10033     sub~matrix~(several~subkeys).
10034 }
10035 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10036 {
10037     Unknown~key.\\
10038     The~key~'\l_keys_key_str'~is~unknown.\\
10039     That~key~will~be~ignored. \\
10040     \c_@@_available_keys_str
10041 }
10042 {
10043     The~available~keys~are~(in~alphabetic~order):~
10044     'delimiters/color',~
10045     'extra~height',~
10046     'hlines',~
10047     'hvlines',~
10048     'left~xshift',~
10049     'name',~
10050     'right~xshift',~
10051     'rules'~(with~the~subkeys~'color'~and~'width'),~
10052     'slim',~
10053     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10054     and~'right~xshift').\\
10055 }
10056 \@@_msg_new:nnn { Unknown~key~for~notes }
10057 {
10058     Unknown~key.\\
10059     The~key~'\l_keys_key_str'~is~unknown.\\
10060     That~key~will~be~ignored. \\
10061     \c_@@_available_keys_str
10062 }
10063 {
10064     The~available~keys~are~(in~alphabetic~order):~
10065     bottomrule,~
10066     code~after,~
10067     code~before,~
10068     detect~duplicates,~
10069     enumitem~keys,~
10070     enumitem~keys~para,~
10071     para,~
10072     label~in~list,~
10073     label~in~tabular~and~
10074     style.
10075 }
10076 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10077 {
10078     Unknown~key.\\
10079     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10080     \token_to_str:N \RowStyle. \\
10081     That~key~will~be~ignored. \\
10082     \c_@@_available_keys_str
10083 }
10084 {
10085     The~available~keys~are~(in~alphabetic~order):~
10086     bold,~
10087     cell~space~top~limit,~
10088     cell~space~bottom~limit,~
10089     cell~space~limits,~
10090     color,~
10091     fill~(alias:~rowcolor),~

```



```

10092     nb-rows,
10093     opacity~and~
10094     rounded-corners.
10095 }
10096 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10097 {
10098     Unknown~key.\\
10099     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10100     \token_to_str:N \NiceMatrixOptions. \\
10101     That~key~will~be~ignored. \\
10102     \c_@@_available_keys_str
10103 }
10104 {
10105     The~available~keys~are~(in~alphabetic~order):~
10106     &-in-blocks,~
10107     allow-duplicate-names,~
10108     ampersand-in-blocks,~
10109     caption-above,~
10110     cell-space-bottom-limit,~
10111     cell-space-limits,~
10112     cell-space-top-limit,~
10113     code-for-first-col,~
10114     code-for-first-row,~
10115     code-for-last-col,~
10116     code-for-last-row,~
10117     corners,~
10118     custom-key,~
10119     create-extra-nodes,~
10120     create-medium-nodes,~
10121     create-large-nodes,~
10122     custom-line,~
10123     delimiters~(several~subkeys),~
10124     end-of-row,~
10125     first-col,~
10126     first-row,~
10127     hlines,~
10128     hvlines,~
10129     hvlines-except-borders,~
10130     last-col,~
10131     last-row,~
10132     left-margin,~
10133     light-syntax,~
10134     light-syntax-expanded,~
10135     matrix/columns-type,~
10136     no-cell-nodes,~
10137     notes~(several~subkeys),~
10138     nullify-dots,~
10139     pgf-node-code,~
10140     renew-dots,~
10141     renew-matrix,~
10142     respect-arraystretch,~
10143     rounded-corners,~
10144     right-margin,~
10145     rules~(with~the~subkeys~'color'~and~'width'),~
10146     small,~
10147     sub-matrix~(several~subkeys),~
10148     vlimes,~
10149     xdots~(several~subkeys).
10150 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10151 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10152 {

```

```

10153     Unknown~key.\\
10154     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10155     \{NiceArray\}. \\
10156     That~key~will~be~ignored. \\
10157     \c_@@_available_keys_str
10158 }
10159 {
10160     The~available~keys~are~(in~alphabetic~order):~
10161     &~in~blocks,~
10162     ampersand~in~blocks,~
10163     b,~
10164     baseline,~
10165     c,~
10166     cell-space-bottom-limit,~
10167     cell-space-limits,~
10168     cell-space-top-limit,~
10169     code-after,~
10170     code-for-first-col,~
10171     code-for-first-row,~
10172     code-for-last-col,~
10173     code-for-last-row,~
10174     columns-width,~
10175     corners,~
10176     create-extra-nodes,~
10177     create-medium-nodes,~
10178     create-large-nodes,~
10179     extra-left-margin,~
10180     extra-right-margin,~
10181     first-col,~
10182     first-row,~
10183     hlines,~
10184     hvlines,~
10185     hvlines-except-borders,~
10186     last-col,~
10187     last-row,~
10188     left-margin,~
10189     light-syntax,~
10190     light-syntax-expanded,~
10191     name,~
10192     no-cell-nodes,~
10193     nullify-dots,~
10194     pgf-node-code,~
10195     renew-dots,~
10196     respect-arraystretch,~
10197     right-margin,~
10198     rounded-corners,~
10199     rules~(with~the~subkeys~'color'~and~'width'),~
10200     small,~
10201     t,~
10202     vlines,~
10203     xdots/color,~
10204     xdots/shorten-start,~
10205     xdots/shorten-end,~
10206     xdots/shorten-and~
10207     xdots/line-style.
10208 }

```

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`).

```

10209 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10210 {
10211     Unknown~key.\\
10212     The~key~'\l_keys_key_str'~is~unknown~for~the~
10213     \@@_full_name_env:. \\

```

```

10214     That~key~will~be~ignored. \\
10215     \c_@@_available_keys_str
10216 }
10217 {
10218     The~available~keys~are~(in~alphabetic~order):~
10219     &~in~blocks,~
10220     ampersand~in~blocks,~
10221     b,~
10222     baseline,~
10223     c,~
10224     cell-space-bottom-limit,~
10225     cell-space-limits,~
10226     cell-space-top-limit,~
10227     code~after,~
10228     code~for~first~col,~
10229     code~for~first~row,~
10230     code~for~last~col,~
10231     code~for~last~row,~
10232     columns~type,~
10233     columns~width,~
10234     corners,~
10235     create~extra~nodes,~
10236     create~medium~nodes,~
10237     create~large~nodes,~
10238     extra~left~margin,~
10239     extra~right~margin,~
10240     first~col,~
10241     first~row,~
10242     hlines,~
10243     hvlines,~
10244     hvlines~except~borders,~
10245     l,~
10246     last~col,~
10247     last~row,~
10248     left~margin,~
10249     light~syntax,~
10250     light~syntax~expanded,~
10251     name,~
10252     no~cell~nodes,~
10253     nullify~dots,~
10254     pgf~node~code,~
10255     r,~
10256     renew~dots,~
10257     respect~arraystretch,~
10258     right~margin,~
10259     rounded~corners,~
10260     rules~(with~the~subkeys~'color'~and~'width'),~
10261     small,~
10262     t,~
10263     vlines,~
10264     xdots/color,~
10265     xdots/shorten~start,~
10266     xdots/shorten~end,~
10267     xdots/shorten~and~
10268     xdots/line~style.
10269 }
10270 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10271 {
10272     Unknown~key.\\
10273     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10274     \{NiceTabular\}. \\
10275     That~key~will~be~ignored. \\
10276     \c_@@_available_keys_str

```

```

10277 }
10278 {
10279   The-available-keys-are~(in~alphabetic~order):~
10280   &-in-blocks,~
10281   ampersand-in-blocks,~
10282   b,~
10283   baseline,~
10284   c,~
10285   caption,~
10286   cell-space-bottom-limit,~
10287   cell-space-limits,~
10288   cell-space-top-limit,~
10289   code-after,~
10290   code-for-first-col,~
10291   code-for-first-row,~
10292   code-for-last-col,~
10293   code-for-last-row,~
10294   columns-width,~
10295   corners,~
10296   custom-line,~
10297   create-extra-nodes,~
10298   create-medium-nodes,~
10299   create-large-nodes,~
10300   extra-left-margin,~
10301   extra-right-margin,~
10302   first-col,~
10303   first-row,~
10304   hlines,~
10305   hvlines,~
10306   hvlines-except-borders,~
10307   label,~
10308   last-col,~
10309   last-row,~
10310   left-margin,~
10311   light-syntax,~
10312   light-syntax-expanded,~
10313   name,~
10314   no-cell-nodes,~
10315   notes~(several~subkeys),~
10316   nullify-dots,~
10317   pgf-node-code,~
10318   renew-dots,~
10319   respect-arraystretch,~
10320   right-margin,~
10321   rounded-corners,~
10322   rules~(with~the~subkeys~'color'~and~'width'),~
10323   short-caption,~
10324   t,~
10325   tabularnote,~
10326   vlides,~
10327   xdots/color,~
10328   xdots/shorten-start,~
10329   xdots/shorten-end,~
10330   xdots/shorten-and~
10331   xdots/line-style.
10332 }
10333 \@@_msg_new:nnn { Duplicate-name }
10334 {
10335   Duplicate-name.\\
10336   The-name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10337   the~same~environment~name~twice.~You~can~go~on,~but,~
10338   maybe,~you~will~have~incorrect~results~especially~
10339   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~

```

```

10340     message~again,~use~the~key~'allow~duplicate~names'~in~
10341     '\token_to_str:N \NiceMatrixOptions'.\\
10342     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10343     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10344   }
10345   {
10346     The~names~already~defined~in~this~document~are:~
10347     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10348   }

10349 \@@_msg_new:nn { Option~auto~for~columns~width }
10350 {
10351   Erroneous~use.\\
10352   You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10353   That~key~will~be~ignored.
10354 }

10355 \@@_msg_new:nn { NiceTabularX~without~X }
10356 {
10357   NiceTabularX~without~X.\\
10358   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10359   However,~you~can~go~on.
10360 }

10361 \@@_msg_new:nn { Preamble~forgotten }
10362 {
10363   Preamble~forgotten.\\
10364   You~have~probably~forgotten~the~preamble~of~your~
10365   \@@_full_name_env:. \\
10366   This~error~is~fatal.
10367 }

10368 \@@_msg_new:nn { Invalid~col~number }
10369 {
10370   Invalid~column~number.\\
10371   A~color~instruction~the~\token_to_str:N \CodeBefore\
10372   specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10373 }

10374 \@@_msg_new:nn { Invalid~row~number }
10375 {
10376   Invalid~row~number.\\
10377   A~color~instruction~the~\token_to_str:N \CodeBefore\
10378   specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10379 }

```

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	130
22	The vertical and horizontal rules	142
23	The empty corners	158
24	The environment <code>{NiceMatrixBlock}</code>	161
25	The extra nodes	162
26	The blocks	166
27	How to draw the dotted lines transparently	190
28	Automatic arrays	191
29	The redefinition of the command <code>\dotfill</code>	192
30	The command <code>\diagbox</code>	192

31	The keyword <code>\CodeAfter</code>	194
32	The delimiters in the preamble	195
33	The command <code>\SubMatrix</code>	196
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	204
35	The command <code>TikzEveryCell</code>	207
36	The command <code>\ShowCellNames</code>	209
37	We process the options at package loading	210
38	About the package underscore	212
39	Error messages of the package	212