

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 24, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

```

212     \skip_horizontal:N \c_zero_dim
213 }

```

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

218 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

282 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
297 \dim_new:N \l_@@_col_width_dim
298 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
299 \int_new:N \g_@@_row_total_int
300 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
303 \tl_new:N \l_@@_hpos_cell_tl
304 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
306 \dim_new:N \g_@@_blocks_ht_dim
307 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We will write in `\g_@@_aux_tl` all the instructions that we have to write on the `aux` file for the current environment. The 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 }`).

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
335 {
```

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

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

```
338   \fi:
```

```
339 }
```

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

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

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

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

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

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

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

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

```

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

346 \bool_new:N \l_@@_bold_row_style_bool

```

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

```

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

```

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

```

353 \tl_new:N \l_@@_code_tl

```

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

```

354 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

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

```

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

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

```

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

```

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

```

359 \bool_new:N \l_@@_in_code_after_bool

```

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

```

360 \bool_new:N \l_@@_ampersand_bool

```

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

```

361 \int_new:N \l_@@_old_iRow_int
362 \int_new:N \l_@@_old_jCol_int

```

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

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

```

363 \seq_new:N \l_@@_custom_line_commands_seq

```

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

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

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

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

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

```
366 \bool_new:N \l_@@_X_columns_aux_bool
367 \dim_new:N \l_@@_X_columns_dim
```

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

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

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

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

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

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

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

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

```
371 \tl_new:N \l_@@_code_before_tl
372 \bool_new:N \l_@@_code_before_bool
```

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

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

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

```
374 \dim_new:N \l_@@_x_initial_dim
375 \dim_new:N \l_@@_y_initial_dim
376 \dim_new:N \l_@@_x_final_dim
377 \dim_new:N \l_@@_y_final_dim
```

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

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

```

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

```

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

```

386 \bool_new:N \g_@@_empty_cell_bool

```

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

```

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

```

The following sequence will contain the characteristics of the blocks of the array, specified by the command `\Block`. Each block is represented by 6 components surrounded by curly braces:

`{imin}{jmin}{imax}{jmax}{options}{contents}`.

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

```

389 \seq_new:N \g_@@_blocks_seq

```

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

```

390 \seq_new:N \g_@@_pos_of_blocks_seq

```

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

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

```

391 \seq_new:N \g_@@_pos_of_xdots_seq

```

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

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

```

392 \seq_new:N \g_@@_pos_of_stroken_blocks_seq

```

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

```

393 \clist_new:N \l_@@_corners_cells_clist

```

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

```

394 \seq_new:N \g_@@_submatrix_names_seq

```

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

```

395 \bool_new:N \l_@@_width_used_bool

```

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

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

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

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

The following counters will be used when drawing the rules.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

423 \bool_new:N \l_@@_nocolor_used_bool

```

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

```

424 \str_new:N \l_@@_vpos_block_str

```

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

```

425 \bool_new:N \l_@@_draw_first_bool

```

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

```

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

```

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

```

428 \int_new:N \g_@@_block_box_int

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

```

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

```

436 \bool_new:N \l_@@_hvlines_bool

```

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

```

437 \bool_new:N \l_@@_dotted_bool

```

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

```

438 \bool_new:N \l_@@_in_caption_bool

```

Variables for the exterior rows and columns

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

• First row

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

```

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

```


- **First column**

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

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

- **Last row**

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

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

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

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

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

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

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

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

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

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

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

Some utilities

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The following internal parameters are for:

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

```

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

```

5 The command `\tabularnote`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

```

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

```

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

```

589     \hbox_set:Nn \l_tmpa_box
590     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

615 {
616   \bool_if:NTF \g_@@_caption_finished_bool
617   {
618     \int_compare:nNtT \c@tabulernote = \g_@@_notes_caption_int
619     { \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`!

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

```

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

```

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

```

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

```

7 The options

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

```

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

```

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

```

698 \bool_new:N \l_@@_caption_above_bool

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

990 \keys_define:nn { nicematrix }

```

```

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

```

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

```

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

```

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


```

1048   renew-matrix .code:n = \@@_renew_matrix: ,
1049   renew-matrix .value_forbidden:n = true ,

```

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

```

1050   exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

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

```

1051   columns-width .code:n =

```

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

```

1052   \str_if_eq:eeTF { #1 } { auto }
1053   { \@@_error:n { Option~auto~for~columns-width } }
1054   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

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

```

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

```

1068 \NewDocumentCommand \NiceMatrixOptions { m }
1069 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }

```

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

```

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

```

```

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

```

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

```

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

```

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

```

1094     small .bool_set:N = \l_@@_small_bool ,
1095     small .value_forbidden:n = true ,
1096     last-col .code:n = \tl_if_empty:nF { #1 }
1097         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1098         \int_zero:N \l_@@_last_col_int ,
1099     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1100     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1101     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1102 }
1103 \keys_define:nn { nicematrix / pNiceArray }
1104 {
1105     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1106     last-col .code:n = \tl_if_empty:nF { #1 }
1107         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1108         \int_zero:N \l_@@_last_col_int ,
1109     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1110     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1111     delimiters / color .value_required:n = true ,
1112     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1113     delimiters / max-width .default:n = true ,
1114     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1115     delimiters .value_required:n = true ,
1116     small .bool_set:N = \l_@@_small_bool ,
1117     small .value_forbidden:n = true ,
1118     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1119     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1120     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1121 }

```

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

8 Important code used by {NiceArrayWithDelims}

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

```

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

```

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

```
1156     \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1158     \int_gincr:N \c@jCol
```

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

```

1159     \int_compare:nNnT \c@jCol = \c_one_int
1160     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }

```

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

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

The following command is nullified in the tabulars.

```

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

```

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

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

We will use a version a little more efficient.

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

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

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

We will use a version a little more efficient.

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

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

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

The following commands are nullified in the tabulars.

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

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

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

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

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

```

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

```

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

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

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1288   \@@_update_max_cell_width:

```

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

```

1289   \@@_update_for_first_and_last_row:

```

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

For example, for the following matrix,

```

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

```

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

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).

```

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

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

```

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

```

1406 \renewcommand{\@tabarray}
1407 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1408 ]

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

We will issue an error only during the first run.

```

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

```

```

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

```

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

overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁴.

```

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

```

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

```

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

```

```

1530 \cs_new_protected:Npn \@@_pre_array_ii:
1531 {

```

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

```

1532   \int_gzero:N \g_@@_total_X_weight_int
1533   \@@_expand_clist:N \l_@@_hlines_clist
1534   \@@_expand_clist:N \l_@@_vlines_clist
1535   \@@_patch_booktabs:
1536   \box_clear_new:N \l_@@_cell_box
1537   \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}`).

```

1538   \bool_if:NT \l_@@_small_bool
1539   {
1540     \cs_set_nopar:Npn \arraystretch { 0.47 }
1541     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1542     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1543   }

1544   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1545   {
1546     \tl_put_right:Nn \@@_begin_of_row:

```

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}`.

```

1547     {
1548         \pgfsys@markposition
1549         { \c@_env: - row - \int_use:N \c@iRow - base }
1550     }
1551 }

```

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

```

1552 \bool_if:NTF \c_@@_tagging_array_bool
1553 {
1554     \cs_set_nopar:Npn \ar@ialign
1555     {
1556         \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1557         \@@_redefine_everycr:
1558         \dim_zero:N \tabskip
1559         \@@_some_initialization:

```

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

```

1560         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1561         \halign
1562     }
1563 }

```

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

```

1564 {
1565     \cs_set_nopar:Npn \ialign
1566     {
1567         \@@_redefine_everycr:
1568         \dim_zero:N \tabskip
1569         \@@_some_initialization:
1570         \cs_set_eq:NN \ialign \@@_old_ialign:
1571         \halign
1572     }
1573 }

```

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).

```

1574 \cs_set_eq:NN \@@_old_ldots \ldots
1575 \cs_set_eq:NN \@@_old_cdots \cdots
1576 \cs_set_eq:NN \@@_old_vdots \vdots
1577 \cs_set_eq:NN \@@_old_ddots \ddots
1578 \cs_set_eq:NN \@@_old_iddots \iddots
1579 \bool_if:NTF \l_@@_standard_cline_bool
1580 { \cs_set_eq:NN \cline \@@_standard_cline }
1581 { \cs_set_eq:NN \cline \@@_cline }
1582 \cs_set_eq:NN \Ldots \@@_Ldots
1583 \cs_set_eq:NN \Cdots \@@_Cdots
1584 \cs_set_eq:NN \Vdots \@@_Vdots
1585 \cs_set_eq:NN \Ddots \@@_Ddots
1586 \cs_set_eq:NN \Iddots \@@_Iddots
1587 \cs_set_eq:NN \Hline \@@_Hline:
1588 \cs_set_eq:NN \Hspace \@@_Hspace:
1589 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1590 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1591 \cs_set_eq:NN \Block \@@_Block:
1592 \cs_set_eq:NN \rotate \@@_rotate:
1593 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1594 \cs_set_eq:NN \dotfill \@@_dotfill:
1595 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:

```

```

1596 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1597 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1598 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1599 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1600 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1601 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1602 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1603 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1604 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1605 \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1606 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1607 \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1608 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1609 \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:`.

```

1610 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1611 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1612 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1613 \@@_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!).

```

1614 \tl_if_exist:NT \l_@@_note_in_caption_tl
1615 {
1616   \tl_if_empty:NF \l_@@_note_in_caption_tl
1617   {
1618     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1619     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1620   }
1621 }

```

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).

```

1622 \seq_gclear:N \g_@@_multicolumn_cells_seq
1623 \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).

```

1624 \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`).

```

1625 \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.

```

1626 \int_gzero_new:N \g_@@_col_total_int
1627 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1628 \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.

```

1629 \tl_gclear_new:N \g_@@_Cdots_lines_tl

```

```

1630 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1631 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1632 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1633 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1634 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1635 \tl_gclear:N \g_nicematrix_code_before_tl
1636 \tl_gclear:N \g_@@_pre_code_before_tl
1637 }

```

This is the end of \@@_pre_array_ii:.

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

```

1638 \cs_new_protected:Npn \@@_pre_array:
1639 {
1640   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1641   \int_gzero_new:N \c@iRow
1642   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1643   \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).

```

1644 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1645 {
1646   \bool_set_true:N \l_@@_last_row_without_value_bool
1647   \bool_if:NT \g_@@_aux_found_bool
1648     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1649 }
1650 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1651 {
1652   \bool_if:NT \g_@@_aux_found_bool
1653     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1654 }

```

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”.

```

1655 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1656 {
1657   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1658   {
1659     \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1660     { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1661     \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1662     { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1663   }
1664 }

1665 \seq_gclear:N \g_@@_cols_vlism_seq
1666 \seq_gclear:N \g_@@_submatrix_seq

```

Now the \CodeBefore.

```

1667 \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.

```

1668 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```
1669 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1670 \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.

```
1671 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1672 \@@_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.

```
1673 \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]`).

```
1674 \dim_zero_new:N \l_@@_left_delim_dim
1675 \dim_zero_new:N \l_@@_right_delim_dim
1676 \bool_if:NTF \g_@@_delims_bool
1677 {
```

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

```
1678 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1679 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1680 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1681 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1682 }
1683 {
1684 \dim_gset:Nn \l_@@_left_delim_dim
1685 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1686 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1687 }
```

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).

```
1688 \hbox_set:Nw \l_@@_the_array_box
1689 \bool_if:NT \c_@@_testphase_table_bool
1690 { \UseTaggingSocket { tbl / hmode / begin } }
1691 \skip_horizontal:N \l_@@_left_margin_dim
1692 \skip_horizontal:N \l_@@_extra_left_margin_dim
1693 \c_math_toggle_token
1694 \bool_if:NTF \l_@@_light_syntax_bool
1695 { \use:c { @@-light-syntax } }
1696 { \use:c { @@-normal-syntax } }
1697 }
```

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

```
1698 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1699 {
1700 \tl_set:Nn \l_tmpa_tl { #1 }
1701 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1702 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1703 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1704 \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...

```
1705 \@@_pre_array:
1706 }
```

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).

```
1707 \cs_new_protected:Npn \@@_pre_code_before:
1708 {
```

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).

```
1709 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1710 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1711 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1712 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
```

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

```
1713 \pgfsys@markposition { \@@_env: - position }
1714 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1715 \pgfpicture
1716 \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1717 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1718 {
1719 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1720 \pgfcoordinate { \@@_env: - row - ##1 }
1721 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1722 }
```

Now, the recreation of the `col` nodes.

```
1723 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1724 {
1725 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1726 \pgfcoordinate { \@@_env: - col - ##1 }
1727 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1728 }
```

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

```
1729 \@@_create_diag_nodes:
```

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

```
1730 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1731 \endpgfpicture
```

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

```
1732 \@@_create_blocks_nodes:
```

```

1733 \IfPackageLoadedT { tikz }
1734 {
1735     \tikzset
1736     {
1737         every-picture / .style =
1738         { overlay , name-prefix = \@@_env: - }
1739     }
1740 }
1741 \cs_set_eq:NN \cellcolor \@@_cellcolor
1742 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1743 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1744 \cs_set_eq:NN \rowcolor \@@_rowcolor
1745 \cs_set_eq:NN \rowcolors \@@_rowcolors
1746 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1747 \cs_set_eq:NN \arraycolor \@@_arraycolor
1748 \cs_set_eq:NN \columncolor \@@_columncolor
1749 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1750 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1751 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1752 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1753 }

```

```

1754 \cs_new_protected:Npn \@@_exec_code_before:
1755 {

```

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...

```

1756 \clist_map_inline:Nn \l_@@_corners_cells_clist
1757 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1758 \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`.

```

1759 \@@_add_to_colors_seq:nn { { nocolor } } { }
1760 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1761 \group_begin:

```

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

```

1762 \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`).

```

1763 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1764 { \@@_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`.

```

1765 \exp_last_unbraced:No \@@_CodeBefore_keys:
1766 \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.

```

1767 \@@_actually_color:
1768 \l_@@_code_before_tl
1769 \q_stop

```

```

1770 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1771 \group_end:
1772 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1773 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1774 }

1775 \keys_define:nn { nicematrix / CodeBefore }
1776 {
1777   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1778   create-cell-nodes .default:n = true ,
1779   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1780   sub-matrix .value_required:n = true ,
1781   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1782   delimiters / color .value_required:n = true ,
1783   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1784 }

1785 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1786 {
1787   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1788   \@@_CodeBefore:w
1789 }

```

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.

```

1790 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1791 {
1792   \bool_if:NT \g_@@_aux_found_bool
1793   {
1794     \@@_pre_code_before:
1795     #1
1796   }
1797 }

```

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.

```

1798 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1799 {
1800   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1801   {
1802     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1803     \pgfcoordinate { \@@_env: - row - ##1 - base }
1804     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1805     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1806     {
1807       \cs_if_exist:cT
1808       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1809       {
1810         \pgfsys@getposition
1811         { \@@_env: - ##1 - #####1 - NW }
1812         \@@_node_position:
1813         \pgfsys@getposition
1814         { \@@_env: - ##1 - #####1 - SE }
1815         \@@_node_position_i:
1816         \@@_pgf_rect_node:nnn
1817         { \@@_env: - ##1 - #####1 }
1818         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1819         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1820       }
1821     }
1822 }

```

```

1822     }
1823     \int_step_inline:nn \c@iRow
1824     {
1825         \pgfnodealias
1826         { \@@_env: - ##1 - last }
1827         { \@@_env: - ##1 - \int_use:N \c@jCol }
1828     }
1829     \int_step_inline:nn \c@jCol
1830     {
1831         \pgfnodealias
1832         { \@@_env: - last - ##1 }
1833         { \@@_env: - \int_use:N \c@iRow - ##1 }
1834     }
1835     \@@_create_extra_nodes:
1836 }

```

```

1837 \cs_new_protected:Npn \@@_create_blocks_nodes:
1838 {
1839     \pgfpicture
1840     \pgf@relevantforpicturesizefalse
1841     \pgfrememberpicturepositiononpagetrue
1842     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1843     { \@@_create_one_block_node:nnnnn ##1 }
1844     \endpgfpicture
1845 }

```

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

```

1846 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1847 {
1848     \tl_if_empty:nF { #5 }
1849     {
1850         \@@_qpoint:n { col - #2 }
1851         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1852         \@@_qpoint:n { #1 }
1853         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1854         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1855         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1856         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1857         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1858         \@@_pgf_rect_node:nnnnn
1859         { \@@_env: - #5 }
1860         { \dim_use:N \l_tmpa_dim }
1861         { \dim_use:N \l_tmpb_dim }
1862         { \dim_use:N \l_@@_tmpc_dim }
1863         { \dim_use:N \l_@@_tmpd_dim }
1864     }
1865 }

```

```

1866 \cs_new_protected:Npn \@@_patch_for_revtext:
1867 {
1868     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1869     \cs_set_eq:NN \insert@column \insert@column@array
1870     \cs_set_eq:NN \@classx \@classx@array
1871     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1872     \cs_set_eq:NN \@arraycr \@arraycr@array
1873     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1874     \cs_set_eq:NN \array \array@array

```

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).

```

1875 \cs_set_eq:NN \@array \@array@array
1876 \cs_set_eq:NN \@tabular \@tabular@array
1877 \cs_set_eq:NN \@mkpream \@mkpream@array
1878 \cs_set_eq:NN \endarray \endarray@array
1879 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1880 \cs_set:Npn \endtabular { \endarray $\egroup} % $
1881 }

```

10 The environment `{NiceArrayWithDelims}`

```

1882 \NewDocumentEnvironment { NiceArrayWithDelims }
1883 { m m O { } m ! O { } t \CodeBefore }
1884 {
1885 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1886 \@@_provide_pgfsyspdfmark:
1887 \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.

```

1888 \bgroup

1889 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1890 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1891 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1892 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1893 \int_gzero:N \g_@@_block_box_int
1894 \dim_zero:N \g_@@_width_last_col_dim
1895 \dim_zero:N \g_@@_width_first_col_dim
1896 \bool_gset_false:N \g_@@_row_of_col_done_bool
1897 \str_if_empty:NT \g_@@_name_env_str
1898 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1899 \bool_if:NTF \l_@@_tabular_bool
1900 \mode_leave_vertical:
1901 \@@_test_if_math_mode:
1902 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1903 \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.

```

1904 \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.

```

1905 \cs_if_exist:NT \tikz@library@external@loaded
1906 {
1907 \tikzexternaldisable
1908 \cs_if_exist:NT \ifstandalone
1909 { \tikzset { external / optimize = false } }
1910 }

```

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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1911 \int_gincr:N \g_@@_env_int
1912 \bool_if:NF \l_@@_block_auto_columns_width_bool
1913 { \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`).

```

1914 \seq_gclear:N \g_@@_blocks_seq
1915 \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`.

```

1916 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1917 \seq_gclear:N \g_@@_pos_of_xdots_seq
1918 \tl_gclear_new:N \g_@@_code_before_tl
1919 \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.

```

1920 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1921 {
1922   \bool_gset_true:N \g_@@_aux_found_bool
1923   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1924 }
1925 { \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.

```

1926 \tl_gclear:N \g_@@_aux_tl
1927 \tl_if_empty:NF \g_@@_code_before_tl
1928 {
1929   \bool_set_true:N \l_@@_code_before_bool
1930   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1931 }
1932 \tl_if_empty:NF \g_@@_pre_code_before_tl
1933 { \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`.

```

1934 \bool_if:NTF \g_@@_delims_bool
1935 { \keys_set:nn { nicematrix / pNiceArray } }
1936 { \keys_set:nn { nicematrix / NiceArray } }
1937 { #3 , #5 }

```

```

1938 \@@_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:.`

```

1939 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1940 }

```

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

```

1941 {
1942   \bool_if:NTF \l_@@_light_syntax_bool
1943   { \use:c { end @@-light-syntax } }
1944   { \use:c { end @@-normal-syntax } }
1945   \c_math_toggle_token
1946   \skip_horizontal:N \l_@@_right_margin_dim
1947   \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

1948
1949 % awful workaround
1950 \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1951 {
1952   \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1953   {
1954     \skip_horizontal:N - \l_@@_columns_width_dim
1955     \bool_if:NTF \l_@@_tabular_bool
1956       { \skip_horizontal:n { - 2 \tabcolsep } }
1957       { \skip_horizontal:n { - 2 \arraycolsep } }
1958   }
1959 }
1960 \hbox_set_end:

```

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

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

```

1961 \bool_if:NT \l_@@_width_used_bool
1962 {
1963   \int_if_zero:nT \g_@@_total_X_weight_int
1964   { \@@_error_or_warning:n { width~without~X~columns } }
1965 }

```

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 .

```

1966 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1967 {
1968   \tl_gput_right:Ne \g_@@_aux_tl
1969   {
1970     \bool_set_true:N \l_@@_X_columns_aux_bool
1971     \dim_set:Nn \l_@@_X_columns_dim
1972     {
1973       \dim_compare:nNnTF
1974       {
1975         \dim_abs:n
1976         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1977       }
1978       <
1979       { 0.001 pt }
1980       { \dim_use:N \l_@@_X_columns_dim }
1981       {
1982         \dim_eval:n
1983         {
1984           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1985           / \int_use:N \g_@@_total_X_weight_int
1986           + \l_@@_X_columns_dim
1987         }
1988       }
1989     }
1990   }
1991 }

```

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).

```

1992 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1993 {
1994   \bool_if:NF \l_@@_last_row_without_value_bool
1995   {
1996     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1997     {
1998       \@@_error:n { Wrong-last-row }
1999       \int_gset_eq:NN \l_@@_last_row_int \c_iRow

```

```

2000     }
2001   }
2002 }

```

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

```

2003   \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2004   \bool_if:NTF \g_@@_last_col_found_bool
2005     { \int_gdecr:N \c@jCol }
2006     {
2007       \int_compare:nNnT \l_@@_last_col_int > { -1 }
2008       { \@@_error:n { last~col~not~used } }
2009     }

```

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

```

2010   \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2011   \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2012   \int_if_zero:nT \l_@@_first_col_int
2013     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2014   \bool_if:nTF { ! \g_@@_delims_bool }
2015     {
2016       \str_if_eq:eeTF \l_@@_baseline_tl { c }
2017       \@@_use_arraybox_with_notes_c:
2018       {
2019         \str_if_eq:eeTF \l_@@_baseline_tl { b }
2020         \@@_use_arraybox_with_notes_b:
2021         \@@_use_arraybox_with_notes:
2022       }
2023     }

```

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).

```

2024   {
2025     \int_if_zero:nTF \l_@@_first_row_int
2026       {
2027         \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2028         \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2029       }
2030     { \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

```

2031     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2032       {
2033         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2034         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2035       }
2036     { \dim_zero:N \l_tmpb_dim }
2037   \hbox_set:Nn \l_tmpa_box
2038     {
2039       \c_math_toggle_token
2040       \@@_color:o \l_@@_delimiters_color_tl
2041       \exp_after:wN \left \g_@@_left_delim_tl
2042       \vcenter

```

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

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

2043 {

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.

```
2044 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2045 \hbox
2046 {
2047   \bool_if:NTF \l_@@_tabular_bool
2048     { \skip_horizontal:N -\tabcolsep }
2049     { \skip_horizontal:N -\arraycolsep }
2050   \@@_use_arraybox_with_notes_c:
2051   \bool_if:NTF \l_@@_tabular_bool
2052     { \skip_horizontal:N -\tabcolsep }
2053     { \skip_horizontal:N -\arraycolsep }
2054 }
```

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

```
2055 \skip_vertical:N -\l_tmpb_dim
2056 \skip_vertical:N \arrayrulewidth
2057 }
2058 \exp_after:wN \right \g_@@_right_delim_tl
2059 \c_math_toggle_token
2060 }
```

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.

```
2061 \bool_if:NTF \l_@@_delimiters_max_width_bool
2062 {
2063   \@@_put_box_in_flow_bis:nn
2064   \g_@@_left_delim_tl
2065   \g_@@_right_delim_tl
2066 }
2067 \@@_put_box_in_flow:
2068 }
```

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).

```
2069 \bool_if:NT \g_@@_last_col_found_bool
2070 { \skip_horizontal:N \g_@@_width_last_col_dim }
2071 \bool_if:NT \l_@@_preamble_bool
2072 {
2073   \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2074     { \@@_warning_gredirect_none:n { columns-not-used } }
2075 }
2076 \@@_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.

2077 \egroup

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

```
2078 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2079 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2080 \iow_now:Ne \@mainaux
2081 {
2082   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2083   { \exp_not:o \g_@@_aux_tl }
2084 }
2085 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2086 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2087 }
```

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

11 We construct the preamble of the array

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

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

```

2088 \cs_new_protected:Npn \@@_transform_preamble:
2089 {
2090   \@@_transform_preamble_i:
2091   \@@_transform_preamble_ii:
2092 }
2093 \cs_new_protected:Npn \@@_transform_preamble_i:
2094 {
2095   \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`).

```

2096   \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2097   \bool_gset_false:N \g_tmpb_bool

```

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

```

2098   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2099   \int_zero:N \l_tmpa_int
2100   \tl_gclear:N \g_@@_array_preamble_tl
2101   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2102   {
2103     \tl_gset:Nn \g_@@_array_preamble_tl
2104     { ! { \skip_horizontal:N \arrayrulewidth } }
2105   }
2106   {
2107     \clist_if_in:NnT \l_@@_vlines_clist 1
2108     {
2109       \tl_gset:Nn \g_@@_array_preamble_tl
2110       { ! { \skip_horizontal:N \arrayrulewidth } }
2111     }
2112   }

```

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

```

2113   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2114   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
2115   \@@_replace_columncolor:
2116 }

```

```

2117 \hook_gput_code:nnn { begindocument } { . }
2118 {
2119   \IfPackageLoadedTF { colortbl }
2120   {

```

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).

```

2121 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2122 \cs_new_protected:Npn \@@_replace_columncolor:
2123 {
2124   \regex_replace_all:NnN
2125     \c_@@_columncolor_regex
2126     { \c { @@_columncolor_preamble } }
2127     \g_@@_array_preamble_tl
2128 }
2129 }
2130 {
2131   \cs_new_protected:Npn \@@_replace_columncolor:
2132     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2133 }
2134 }

```

```

2135 \cs_new_protected:Npn \@@_transform_preamble_ii:
2136 {

```

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

```

2137 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2138 {
2139   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2140     { \bool_gset_true:N \g_@@_delims_bool }
2141 }
2142 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2143 \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).

```

2144 \int_if_zero:nTF \l_@@_first_col_int
2145 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2146 {
2147   \bool_if:NF \g_@@_delims_bool
2148   {
2149     \bool_if:NF \l_@@_tabular_bool
2150     {
2151       \clist_if_empty:NT \l_@@_vlines_clist
2152       {
2153         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2154         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2155       }
2156     }
2157   }
2158 }
2159 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2160 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2161 {
2162   \bool_if:NF \g_@@_delims_bool
2163   {
2164     \bool_if:NF \l_@@_tabular_bool
2165     {
2166       \clist_if_empty:NT \l_@@_vlines_clist
2167       {
2168         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2169         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2170       }
2171     }
2172   }

```

```

2171     }
2172   }
2173 }

```

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`).

```

2174   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2175   {
2176     \tl_gput_right:Nn \g_@@_array_preamble_tl
2177     { > { \@@_error_too_much_cols: } 1 }
2178   }
2179 }

```

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.

```

2180 \cs_new_protected:Npn \@@_rec_preamble:n #1
2181 {

```

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

```

2182   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2183   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2184   {

```

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

```

2185     \cs_if_exist:cTF { NC @ find @ #1 }
2186     {
2187       \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2188       \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2189     }
2190     {
2191       \str_if_eq:nnTF { #1 } { S }
2192       { \@@_fatal:n { unknown~column~type~S } }
2193       { \@@_fatal:nn { unknown~column~type } { #1 } }
2194     }
2195   }
2196 }

```

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

```

2197 \cs_new:Npn \@@_c #1
2198 {
2199   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2200   \tl_gclear:N \g_@@_pre_cell_tl
2201   \tl_gput_right:Nn \g_@@_array_preamble_tl
2202   { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2203   \int_gincr:N \c@jCol
2204   \@@_rec_preamble_after_col:n
2205 }
2206 \cs_new:Npn \@@_l #1
2207 {
2208   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2209   \tl_gclear:N \g_@@_pre_cell_tl
2210   \tl_gput_right:Nn \g_@@_array_preamble_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`.

```

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

```

For ! and @

```

2232 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2233 {
2234   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2235   \@@_rec_preamble:n
2236 }
2237 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2238 \cs_new:cpn { @@ _ | } #1
2239 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2240   \int_incr:N \l_tmpa_int
2241   \@@_make_preamble_i_i:n
2242 }
2243 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2244 {
2245   \str_if_eq:nnTF { #1 } { | }
2246   { \use:c { @@ _ | } | }
2247   { \@@_make_preamble_i_ii:nn { } #1 }
2248 }
2249 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2250 {
2251   \str_if_eq:nnTF { #2 } { [ ] }
2252   { \@@_make_preamble_i_iii:nw { #1 } [ ] }
2253   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2254 }
2255 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2256 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2257 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2258 {
2259   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2260   \tl_gput_right:Ne \g_@@_array_preamble_tl
2261   {

```

Here, the command \dim_eval:n is mandatory.

```

2262   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2263 }
2264 \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

2265     {
2266     \@@_vline:n
2267     {
2268         position = \int_eval:n { \c@jCol + 1 } ,
2269         multiplicity = \int_use:N \l_tmpa_int ,
2270         total-width = \dim_use:N \l_@@_rule_width_dim ,
2271         #2
2272     }

```

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.

```

2273     }
2274     \int_zero:N \l_tmpa_int
2275     \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2276     \@@_rec_preamble:n #1
2277 }

```

```

2278 \cs_new:cpn { @@ _ > } #1 #2
2279 {
2280     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2281     \@@_rec_preamble:n
2282 }
2283 \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.

```

2284 \keys_define:nn { nicematrix / p-column }
2285 {
2286     r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2287     r .value_forbidden:n = true ,
2288     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2289     c .value_forbidden:n = true ,
2290     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2291     l .value_forbidden:n = true ,
2292     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2293     S .value_forbidden:n = true ,
2294     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2295     p .value_forbidden:n = true ,
2296     t .meta:n = p ,
2297     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2298     m .value_forbidden:n = true ,
2299     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2300     b .value_forbidden:n = true
2301 }

```

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

```

2302 \cs_new:Npn \@@_p #1
2303 {
2304     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

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

```

```

2315 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2316 { \@@_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.

```

2317 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2318 {

```

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).

```

2319 \str_set:Nn \l_@@_hpos_col_str { j }
2320 \@@_keys_p_column:n { #1 }
2321 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2322 }
2323 \cs_new_protected:Npn \@@_keys_p_column:n #1
2324 { \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.

```

2325 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2326 {
2327 \use:e
2328 {
2329 \@@_make_preamble_ii_v:nnnnnnnn
2330 { \str_if_eq:nnTF \l_@@_vpos_col_str { p } { t } { b } }
2331 { \dim_eval:n { #1 } }
2332 {

```

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.

```

2333 \str_if_eq:nnTF \l_@@_hpos_col_str { j }
2334 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2335 {

```

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

```

2336 \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2337 { \str_lowercase:o \l_@@_hpos_col_str }
2338 }
2339 \IfPackageLoadedTF { ragged2e }
2340 {
2341 \str_case:on \l_@@_hpos_col_str
2342 {
2343 c { \exp_not:N \Centering }
2344 l { \exp_not:N \RaggedRight }
2345 r { \exp_not:N \RaggedLeft }
2346 }
2347 }
2348 {
2349 \str_case:on \l_@@_hpos_col_str
2350 {
2351 c { \exp_not:N \centering }
2352 l { \exp_not:N \raggedright }
2353 r { \exp_not:N \raggedleft }
2354 }
2355 }
2356 #3
2357 }
2358 { \str_if_eq:nnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2359 { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2360 { \str_if_eq:nnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2361 { #2 }

```

```

2362     {
2363         \str_case:onF \l_@@_hpos_col_str
2364         {
2365             { j } { c }
2366             { si } { c }
2367         }

```

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

```

2368         { \str_lowercase:o \l_@@_hpos_col_str }
2369     }
2370 }

```

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

```

2371     \int_gincr:N \c_jCol
2372     \@@_rec_preamble_after_col:n
2373 }

```

#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*.

```

2374 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2375 {
2376     \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2377     {
2378         \tl_gput_right:Nn \g_@@_array_preamble_tl
2379         { > \@@_test_if_empty_for_S: }
2380     }
2381     {
2382         \tl_gput_right:Nn \g_@@_array_preamble_tl
2383         { > \@@_test_if_empty: }
2384     }
2385     \tl_gput_right:Nn \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2386     \tl_gclear:N \g_@@_pre_cell_tl
2387     \tl_gput_right:Nn \g_@@_array_preamble_tl
2388     {
2389         > {

```

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.

```

2390         \dim_set:Nn \l_@@_col_width_dim { #2 }
2391         \bool_if:NT \c_@@_testphase_table_bool
2392         { \tag_struct_begin:n { tag = Div } }
2393         \@@_cell_begin:

```

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

```

2394         \use:c { #7 } [ #1 ] { #2 }

```

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

```

2395     \everypar
2396     {
2397         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2398         \everypar { }
2399     }
2400     \bool_if:NT \c_@@_testphase_table_bool \tagpdfparaOn

```


Now, the potential code for the horizontal position of the content of the cell (`\centering`, `\raggedright`, `\RaggedRight`, etc.).

```
2401         #3
```

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

```
2402         \g_@@_row_style_tl
2403         \arraybackslash
2404         #5
2405     }
2406     #8
2407     < {
2408         #6
```

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

```
2409         \@finalstrut \@arstrutbox
2410         \use:c { end #7 }
```

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

```
2411         #4
2412         \@@_cell_end:
2413         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2414     }
2415 }
2416 }
```

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

```
2417 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2418 {
```

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 `&`).

```
2419     \group_align_safe_begin:
2420     \peek_meaning:NTF &
2421     {
2422         \group_align_safe_end:
2423         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2424         {
2425             \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2426             \skip_horizontal:N \l_@@_col_width_dim
2427         }
2428     }
2429     { \group_align_safe_end: }
2430 }

2431 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2432 {
2433     \peek_meaning:NT \__siunitx_table_skip:n
2434     {
2435         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2436         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2437     }
2438 }
```

The following command will be used in `m`-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in `array`) that if the height of the cell is no more that the height of `\strutbox`, there is only one row.

```
2439 \cs_new_protected:Npn \@@_center_cell_box:
2440 {
```

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

```

2441 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2442 {
2443   \int_compare:nNnT
2444     { \box_ht:N \l_@@_cell_box }
2445     >

```

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).

```

2446   { \box_ht:N \strutbox }
2447   {
2448     \hbox_set:Nn \l_@@_cell_box
2449       {
2450         \box_move_down:nn
2451           {
2452             ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2453               + \baselineskip ) / 2
2454           }
2455           { \box_use:N \l_@@_cell_box }
2456       }
2457   }
2458 }
2459 }

```

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

```

2460 \cs_new:Npn \@@_V #1 #2
2461 {
2462   \str_if_eq:nnTF { #1 } { [ ] }
2463     { \@@_make_preamble_V_i:w [ ] }
2464     { \@@_make_preamble_V_i:w [ ] { #2 } }
2465 }
2466 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2467 { \@@_make_preamble_V_ii:nn { #1 } }
2468 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2469 {
2470   \str_set:Nn \l_@@_vpos_col_str { p }
2471   \str_set:Nn \l_@@_hpos_col_str { j }
2472   \@@_keys_p_column:n { #1 }
2473   \IfPackageLoadedTF { varwidth }
2474     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2475     {
2476       \@@_error_or_warning:n { varwidth-not-loaded }
2477       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2478     }
2479 }

```

For `w` and `W`

```

2480 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2481 \cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

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

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

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

#4 is the width of the column.

```

2482 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2483 {
2484   \str_if_eq:nnTF { #3 } { s }
2485     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2486     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2487 }

```

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.

```

2488 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2489 {
2490   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2491   \tl_gclear:N \g_@@_pre_cell_tl
2492   \tl_gput_right:Nn \g_@@_array_preamble_tl
2493   {
2494     > {
2495       \dim_set:Nn \l_@@_col_width_dim { #2 }
2496       \@@_cell_begin:
2497       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2498     }
2499     c
2500     < {
2501       \@@_cell_end_for_w_s:
2502       #1
2503       \@@_adjust_size_box:
2504       \box_use_drop:N \l_@@_cell_box
2505     }
2506   }
2507   \int_gincr:N \c@jCol
2508   \@@_rec_preamble_after_col:n
2509 }

```

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

```

2510 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2511 {
2512   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2513   \tl_gclear:N \g_@@_pre_cell_tl
2514   \tl_gput_right:Nn \g_@@_array_preamble_tl
2515   {
2516     > {

```

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.

```

2517       \dim_set:Nn \l_@@_col_width_dim { #4 }
2518       \hbox_set:Nw \l_@@_cell_box
2519       \@@_cell_begin:
2520       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2521     }
2522     c
2523     < {
2524       \@@_cell_end:
2525       \hbox_set_end:
2526       #1
2527       \@@_adjust_size_box:
2528       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2529     }
2530   }

```

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

```

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

```

```

2534 \cs_new_protected:Npn \@@_special_W:
2535 {
2536   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2537   { \@@_warning:n { W~warning } }
2538 }

```

For S (of siunitx).

```

2539 \cs_new:Npn \@@_S #1 #2
2540 {
2541   \str_if_eq:nnTF { #1 } { [ ] }
2542     { \@@_make_preamble_S:w [ ] }
2543     { \@@_make_preamble_S:w [ ] { #2 } }
2544 }
2545 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2546 { \@@_make_preamble_S_i:n { #1 } }
2547 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2548 {
2549   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2550   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2551   \tl_gclear:N \g_@@_pre_cell_tl
2552   \tl_gput_right:Nn \g_@@_array_preamble_tl
2553     {
2554       > {
2555         \@@_cell_begin:
2556         \keys_set:nn { siunitx } { #1 }
2557         \siunitx_cell_begin:w
2558       }
2559       c
2560       < { \siunitx_cell_end: \@@_cell_end: }
2561     }

```

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

```

2562   \int_gincr:N \c@jCol
2563   \@@_rec_preamble_after_col:n
2564 }

```

For (, [and \{.

```

2565 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2566 {
2567   \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.

```

2568   \int_if_zero:nTF \c@jCol
2569   {
2570     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2571     {

```

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

```

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

```

```

2592     \@@_rec_preamble:n
2593   }
2594   { \@@_rec_preamble:n #2 }
2595 }

```

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

```

2596 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2597 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2598 {
2599   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2600   \tl_if_in:nnTF { ) } [ \ ] { #2 }
2601   { \@@_make_preamble_v:nnn #1 #2 }
2602   {
2603     \str_if_eq:nnTF { \@@_stop: } { #2 }
2604     {
2605       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2606       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2607       {
2608         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2609         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2610         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2611         \@@_rec_preamble:n #2
2612       }
2613     }
2614     {
2615       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2616       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2617       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2618       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2619       \@@_rec_preamble:n #2
2620     }
2621   }
2622 }
2623 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2624 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2625 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2626 {
2627   \str_if_eq:nnTF { \@@_stop: } { #3 }
2628   {
2629     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2630     {
2631       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2632       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2633       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2634       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2635     }
2636     {
2637       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2638       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2639       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2640       \@@_error:nn { double~closing~delimiter } { #2 }
2641     }
2642   }
2643   {
2644     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2645     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2646     \@@_error:nn { double~closing~delimiter } { #2 }
2647     \@@_rec_preamble:n #3

```

```

2648     }
2649 }

2650 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2651 { \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 @{...}.

```

2652 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2653 {
2654   \str_if_eq:nnTF { #1 } { < }
2655     \@@_rec_preamble_after_col_i:n
2656     {
2657       \str_if_eq:nnTF { #1 } { @ }
2658         \@@_rec_preamble_after_col_ii:n
2659         {
2660           \str_if_eq:nnTF \l_@@_vlines_clist { all }
2661           {
2662             \tl_gput_right:Nn \g_@@_array_preamble_tl
2663               { ! { \skip_horizontal:N \arrayrulewidth } }
2664           }
2665           {
2666             \clist_if_in:NeT \l_@@_vlines_clist
2667               { \int_eval:n { \c@jCol + 1 } }
2668               {
2669                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2670                   { ! { \skip_horizontal:N \arrayrulewidth } }
2671               }
2672           }
2673           \@@_rec_preamble:n { #1 }
2674         }
2675       }
2676     }
2677 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2678 {
2679   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2680   \@@_rec_preamble_after_col:n
2681 }

```

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.

```

2682 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2683 {
2684   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2685   {
2686     \tl_gput_right:Nn \g_@@_array_preamble_tl
2687       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2688   }
2689   {
2690     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2691     {
2692       \tl_gput_right:Nn \g_@@_array_preamble_tl
2693         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2694     }
2695     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2696   }
2697   \@@_rec_preamble:n
2698 }

2699 \cs_new:cpn { @@ _ * } #1 #2 #3

```

```

2700 {
2701   \tl_clear:N \l_tmpa_tl
2702   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2703   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2704 }

```

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

```

2705 \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }

```

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

```

2706 \cs_new:Npn \@@_X #1 #2
2707 {
2708   \str_if_eq:nnTF { #2 } { [ ]
2709     { \@@_make_preamble_X:w [ ] }
2710     { \@@_make_preamble_X:w [ ] #2 }
2711   }
2712   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2713   { \@@_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`).

```

2714 \keys_define:nn { nicematrix / X-column }
2715 { 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`.

```

2716 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2717 {

```

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`).

```

2718   \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`).

```

2719   \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`.

```

2720   \int_zero_new:N \l_@@_weight_int
2721   \int_set_eq:NN \l_@@_weight_int \c_one_int
2722   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2723   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2724   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2725   {
2726     \@@_error_or_warning:n { negative-weight }
2727     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2728   }
2729   \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).

```

2730 \bool_if:NTF \l_@@_X_columns_aux_bool
2731 {
2732   \@@_make_preamble_ii_iv:nnn
2733   { \l_@@_weight_int \l_@@_X_columns_dim }
2734   { minipage }
2735   { \@@_no_update_width: }
2736 }
2737 {
2738   \tl_gput_right:Nn \g_@@_array_preamble_tl
2739   {
2740     > {
2741       \@@_cell_begin:
2742       \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`.

```

2743 \NotEmpty

```

The following code will nullify the box of the cell.

```

2744 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2745 { \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`.

```

2746 \begin { minipage } { 5 cm } \arraybackslash
2747 {
2748   c
2749   < {
2750     \end { minipage }
2751     \@@_cell_end:
2752   }
2753 }
2754 \int_gincr:N \c@jCol
2755 \@@_rec_preamble_after_col:n
2756 }
2757 }

```

```

2758 \cs_new_protected:Npn \@@_no_update_width:
2759 {
2760   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2761   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2762 }

```

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

```

2763 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2764 {
2765   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2766   { \int_eval:n { \c@jCol + 1 } }
2767   \tl_gput_right:Ne \g_@@_array_preamble_tl
2768   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2769   \@@_rec_preamble:n
2770 }

```

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.

```

2771 \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).

```

2772 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2773 { \@@_fatal:n { Preamble-forgotten } }
2774 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2775 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2776 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```

2777 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2778 {

```

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`.

```

2779 \multispan { #1 }
2780 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2781 \begingroup
2782 \bool_if:NT \c_@@_testphase_table_bool
2783 { \tbl_update_multicolumn_cell_data:n { #1 } }
2784 \cs_set_nopar:Npn \@addamp
2785 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2786 \tl_gclear:N \g_@@_preamble_tl
2787 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2788 \exp_args:No \@mkpream \g_@@_preamble_tl
2789 \@addtopreamble \@empty
2790 \endgroup
2791 \bool_if:NT \c_@@_testphase_table_bool
2792 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2793 \int_compare:nNnT { #1 } > \c_one_int
2794 {
2795   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2796   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } } }
2797   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2798   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2799   {
2800     {
2801       \int_if_zero:nTF \c@jCol
2802       { \int_eval:n { \c@iRow + 1 } }
2803       { \int_use:N \c@iRow }
2804     }
2805     { \int_eval:n { \c@jCol + 1 } } }
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   { } % for the name of the block

```

```

2813     }
2814 }

```

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

```

2815 \RenewDocumentCommand \cellcolor { 0 { } m }
2816 {
2817   \@@_test_color_inside:
2818   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2819   {
2820     \@@_rectanglecolor [ ##1 ]
2821     { \exp_not:n { ##2 } }
2822     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2823     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2824   }
2825   \ignorespaces
2826 }

```

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

```

2827 \cs_set_nopar:Npn \@sharp { #3 }
2828 \@arstrut
2829 \@preamble
2830 \null

```

We add some lines.

```

2831 \int_gadd:Nn \c@jCol { #1 - 1 }
2832 \int_compare:nNt \c@jCol > \g_@@_col_total_int
2833 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2834 \ignorespaces
2835 }

```

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`.

```

2836 \cs_new_protected:Npn \@_make_m_preamble:n #1
2837 {
2838   \str_case:nnF { #1 }
2839   {
2840     c { \@_make_m_preamble_i:n #1 }
2841     l { \@_make_m_preamble_i:n #1 }
2842     r { \@_make_m_preamble_i:n #1 }
2843     > { \@_make_m_preamble_ii:nn #1 }
2844     ! { \@_make_m_preamble_ii:nn #1 }
2845     @ { \@_make_m_preamble_ii:nn #1 }
2846     | { \@_make_m_preamble_iii:n #1 }
2847     p { \@_make_m_preamble_iv:nnn t #1 }
2848     m { \@_make_m_preamble_iv:nnn c #1 }
2849     b { \@_make_m_preamble_iv:nnn b #1 }
2850     w { \@_make_m_preamble_v:nnnn { } #1 }
2851     W { \@_make_m_preamble_v:nnnn { \@_special_W: } #1 }
2852     \q_stop { }
2853   }
2854   {
2855     \cs_if_exist:cTF { NC @ find @ #1 }
2856     {
2857       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2858       \exp_last_unbraced:No \@_make_m_preamble:n \l_tmpa_tl
2859     }
2860     {
2861       \str_if_eq:nnTF { #1 } { S }
2862       { \@_fatal:n { unknown~column~type~S } }
2863       { \@_fatal:nn { unknown~column~type } { #1 } }

```

```

2864     }
2865   }
2866 }

```

For c, l and r

```

2867 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2868 {
2869   \tl_gput_right:Nn \g_@@_preamble_tl
2870   {
2871     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2872     #1
2873     < \@@_cell_end:
2874   }

```

We test for the presence of a <.

```

2875   \@@_make_m_preamble_x:n
2876 }

```

For >, ! and @

```

2877 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2878 {
2879   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2880   \@@_make_m_preamble:n
2881 }

```

For |

```

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

```

For p, m and b

```

2887 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2888 {
2889   \tl_gput_right:Nn \g_@@_preamble_tl
2890   {
2891     > {
2892       \@@_cell_begin:
2893       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2894       \mode_leave_vertical:
2895       \arraybackslash
2896       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2897     }
2898     c
2899     < {
2900       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2901       \end { minipage }
2902       \@@_cell_end:
2903     }
2904   }

```

We test for the presence of a <.

```

2905   \@@_make_m_preamble_x:n
2906 }

```

For w and W

```

2907 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2908 {
2909   \tl_gput_right:Nn \g_@@_preamble_tl
2910   {
2911     > {
2912       \dim_set:Nn \l_@@_col_width_dim { #4 }
2913       \hbox_set:Nw \l_@@_cell_box

```

```

2914         \@@_cell_begin:
2915         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2916     }
2917     c
2918     < {
2919         \@@_cell_end:
2920         \hbox_set_end:
2921         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2922         #1
2923         \@@_adjust_size_box:
2924         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2925     }
2926 }

```

We test for the presence of a <.

```

2927     \@@_make_m_preamble_x:n
2928 }

```

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

```

2929 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2930 {
2931     \str_if_eq:nnTF { #1 } { < }
2932     \@@_make_m_preamble_ix:n
2933     { \@@_make_m_preamble:n { #1 } }
2934 }
2935 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2936 {
2937     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2938     \@@_make_m_preamble_x:n
2939 }

```

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).

```

2940 \cs_new_protected:Npn \@@_put_box_in_flow:
2941 {
2942     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2943     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2944     \str_if_eq:eeTF \l_@@_baseline_tl { c }
2945     { \box_use_drop:N \l_tmpa_box }
2946     \@@_put_box_in_flow_i:
2947 }

```

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

```

2948 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2949 {
2950     \pgfpicture
2951     \@@_qpoint:n { row - 1 }
2952     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2953     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2954     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2955     \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).

```

2956     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2957     {
2958         \int_set:Nn \l_tmpa_int
2959         {
2960             \str_range:Nnn

```

```

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

```

We take into account the position of the mathematical axis.

```

2983         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2984     }
2985     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2986     \endpgfpicture
2987     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2988     \box_use_drop:N \l_tmpa_box
2989 }

```

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).

```

2990 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2991 {

```

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.

```

2992     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2993     {
2994         \int_compare:nNnT \c_jCol > \c_one_int
2995         {
2996             \box_set_wd:Nn \l_@@_the_array_box
2997             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2998         }
2999     }

```

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

```

3000     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3001     \bool_if:NT \l_@@_caption_above_bool
3002     {
3003         \tl_if_empty:NF \l_@@_caption_tl
3004         {
3005             \bool_set_false:N \g_@@_caption_finished_bool
3006             \int_gzero:N \c@tabulernote
3007             \@@_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).

```

3008         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3009         {
3010             \tl_gput_right:Ne \g_@@_aux_tl
3011             {
3012                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3013                 { \int_use:N \g_@@_notes_caption_int }
3014             }
3015             \int_gzero:N \g_@@_notes_caption_int
3016         }
3017     }
3018 }

```

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

```

3019     \hbox
3020     {
3021         \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.

```

3022         \@@_create_extra_nodes:
3023         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3024     }

```

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).

```

3025     \bool_lazy_any:nT
3026     {
3027         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3028         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3029         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3030     }
3031     \@@_insert_tabularnotes:
3032     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3033     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3034     \end { minipage }
3035 }

```

```

3036 \cs_new_protected:Npn \@@_insert_caption:
3037 {
3038     \tl_if_empty:NF \l_@@_caption_tl
3039     {
3040         \cs_if_exist:NTF \@captive
3041         { \@@_insert_caption_i: }
3042         { \@@_error:n { caption~outside~float } }
3043     }
3044 }

```

```

3045 \cs_new_protected:Npn \@@_insert_caption_i:
3046 {
3047     \group_begin:

```

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

```

3048     \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.

```

3049 \IfPackageLoadedT { floatrow }
3050 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3051 \tl_if_empty:NTF \l_@@_short_caption_tl
3052 { \caption }
3053 { \caption [ \l_@@_short_caption_tl ] }
3054 { \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.

```

3055 \bool_if:NF \g_@@_caption_finished_bool
3056 {
3057   \bool_gset_true:N \g_@@_caption_finished_bool
3058   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3059   \int_gzero:N \c@tabularnote
3060 }
3061 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3062 \group_end:
3063 }

3064 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3065 {
3066   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3067   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3068 }

3069 \cs_new_protected:Npn \@@_insert_tabularnotes:
3070 {
3071   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3072   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3073   \skip_vertical:N 0.65ex

```

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

```

3074 \group_begin:
3075 \l_@@_notes_code_before_tl
3076 \tl_if_empty:NF \g_@@_tabularnote_tl
3077 {
3078   \g_@@_tabularnote_tl \par
3079   \tl_gclear:N \g_@@_tabularnote_tl
3080 }

```

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.

```

3081 \int_compare:nNnT \c@tabularnote > \c_zero_int
3082 {
3083   \bool_if:NTF \l_@@_notes_para_bool
3084   {
3085     \begin { tabularnotes* }
3086     \seq_map_inline:Nn \g_@@_notes_seq
3087       { \@_one_tabularnote:nn ##1 }
3088     \strut
3089     \end { tabularnotes* }

```

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

```

3090 \par
3091 }
3092 {
3093   \tabularnotes
3094   \seq_map_inline:Nn \g_@@_notes_seq

```

```

3095         { \@@_one_tabularnote:nn #1 }
3096         \strut
3097     \endtabularnotes
3098 }
3099 }
3100 \unskip
3101 \group_end:
3102 \bool_if:NT \l_@@_notes_bottomrule_bool
3103 {
3104     \IfPackageLoadedTF { booktabs }
3105     {

```

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

```

3106         \skip_vertical:N \aboverulesep

```

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

```

3107         { \CT@arc@ \hrule height \heavyrulewidth }
3108     }
3109     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3110 }
3111 \l_@@_notes_code_after_tl
3112 \seq_gclear:N \g_@@_notes_seq
3113 \seq_gclear:N \g_@@_notes_in_caption_seq
3114 \int_gzero:N \c@tabularnote
3115 }

```

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.

```

3116 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3117 {
3118     \tl_if_novalue:nTF { #1 }
3119     { \item }
3120     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3121 }

```

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`.

```

3122 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3123 {
3124     \pgfpicture
3125     \@@_qpoint:n { row - 1 }
3126     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3127     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3128     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3129     \endpgfpicture
3130     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3131     \int_if_zero:nT \l_@@_first_row_int
3132     {
3133         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3134         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3135     }
3136     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3137 }

```

Now, the general case.

```

3138 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3139 {

```

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

```

3140     \tl_if_eq:NnT \l_@@_baseline_tl { t }
3141     { \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`.

```

3142 \pgfpicture
3143 \@@_qpoint:n { row - 1 }
3144 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3145 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3146 {
3147   \int_set:Nn \l_tmpa_int
3148   {
3149     \str_range:Nnn
3150     \l_@@_baseline_tl
3151     6
3152     { \tl_count:o \l_@@_baseline_tl }
3153   }
3154   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3155 }
3156 {
3157   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3158   \bool_lazy_or:nnT
3159   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3160   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3161   {
3162     \@@_error:n { bad~value~for~baseline }
3163     \int_set:Nn \l_tmpa_int 1
3164   }
3165   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3166 }
3167 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3168 \endpgfpicture
3169 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3170 \int_if_zero:nT \l_@@_first_row_int
3171 {
3172   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3173   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3174 }
3175 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3176 }

```

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.

```

3177 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3178 {

```

We will compute the real width of both delimiters used.

```

3179 \dim_zero_new:N \l_@@_real_left_delim_dim
3180 \dim_zero_new:N \l_@@_real_right_delim_dim
3181 \hbox_set:Nn \l_tmpb_box
3182 {
3183   \c_math_toggle_token
3184   \left #1
3185   \vcenter
3186   {
3187     \vbox_to_ht:nn
3188     { \box_ht_plus_dp:N \l_tmpa_box }
3189     { }
3190   }
3191   \right .
3192   \c_math_toggle_token
3193 }
3194 \dim_set:Nn \l_@@_real_left_delim_dim
3195 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3196 \hbox_set:Nn \l_tmpb_box

```

```

3197 {
3198   \c_math_toggle_token
3199   \left .
3200   \vbox_to_ht:nn
3201     { \box_ht_plus_dp:N \l_tmpa_box }
3202     { }
3203   \right #2
3204   \c_math_toggle_token
3205 }
3206 \dim_set:Nn \l_@@_real_right_delim_dim
3207 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

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

```

The construction of the array in the environment `{NiceArrayWithDelims}` is, in fact, done by the environment `{@@-light-syntax}` or by the environment `{@@-normal-syntax}` (whether the option `light-syntax` is in force or not). When the key `light-syntax` is not used, the construction is a standard environment (and, thus, it's possible to use verbatim in the array).

```

3214 \NewDocumentEnvironment { @@-normal-syntax } { }

```

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

```

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

```

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

```

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

```

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

```

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

```

First, we test whether the environment is empty. It's only a security. Of course, this test is more easy than the similar test for the “normal syntax” because we have the whole body of the environment in `#1`.

```

3232 \tl_if_empty:nT { #1 }
3233 { \@@_fatal:n { empty-environment } }
3234 \tl_if_in:nnT { #1 } { & }
3235 { \@@_fatal:n { ampersand-in-light-syntax } }
3236 \tl_if_in:nnT { #1 } { \ }
3237 { \@@_fatal:n { double-backslash-in-light-syntax } }

```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

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

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

```
3239 }
```

Now, the second part of the environment. We must leave these lines in the second part (and not put them in the first part even though we caught the whole body of the environment with an argument of type b) in order to have the columns `S` of `siunitx` working fine.

```
3240 {
3241 \@@_create_col_nodes:
3242 \endarray
3243 }
3244 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3245 {
3246 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }
```

The body of the array, which is stored in the argument `#1`, is now splitted into items (and *not* tokens).

```
3247 \seq_clear_new:N \l_@@_rows_seq
```

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

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

We delete the last row if it is empty.

```
3253 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3254 \tl_if_empty:NF \l_tmpa_tl
3255 { \seq_put_right:No \l_@@_rows_seq \l_tmpa_tl }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
3256 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3257 { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }
```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`.

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

First, we treat the first row.

```
3260 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3261 \@@_line_with_light_syntax:o \l_tmpa_tl
```

Now, the other rows (with the same treatment, excepted that we have to insert `\\` between the rows).

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

Now, we can construct the preamble: if the user has used the key `last-col`, we have the correct number of columns even though the user has used `last-col` without value.

```
3273 \@@_transform_preamble:
```

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

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

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

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

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

```
3296 \end { #2 }
3297 }
```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns such as `columns-width`).

```
3298 \cs_new:Npn \@@_create_col_nodes:
3299 {
3300   \crrr
3301   \int_if_zero:nT \l_@@_first_col_int
3302   {
3303     \omit
3304     \hbox_overlap_left:n
3305     {
3306       \bool_if:NT \l_@@_code_before_bool
3307       { \pgfsys@markposition { \@@_env: - col - 0 } }
3308       \pgfpicture
3309       \pgfrememberpicturepositiononpagetrue
3310       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3311       \str_if_empty:NF \l_@@_name_str
3312       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3313       \endpgfpicture
3314       \skip_horizontal:N 2\col@sep
3315       \skip_horizontal:N \g_@@_width_first_col_dim
3316     }
```

```

3317      &
3318    }
3319  \omit

```

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

```

3320  \bool_gset_true:N \g_@@_row_of_col_done_bool

```

First, we put a `col` node on the left of the first column (of course, we have to do that *after* the `\omit`).

```

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

```

We compute in `\g_tmpa_skip` the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an `\halign` and because we have to use that variable in other cells (of the same row). The affectation of `\g_tmpa_skip`, like all the affectations, must be done after the `\omit` of the cell.

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but we will add some dimensions to it.

```

3358  \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3359  \bool_if:NF \l_@@_auto_columns_width_bool
3360  { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3361  {
3362    \bool_lazy_and:nnTF
3363    \l_@@_auto_columns_width_bool
3364    { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3365    { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3366    { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3367  \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3368  }

```

```

3369 \skip_horizontal:N \g_tmpa_skip
3370 \hbox
3371 {
3372   \bool_if:NT \l_@@_code_before_bool
3373   {
3374     \hbox
3375     {
3376       \skip_horizontal:N -0.5\arrayrulewidth
3377       \pgfsys@markposition { \@@_env: - col - 2 }
3378       \skip_horizontal:N 0.5\arrayrulewidth
3379     }
3380   }
3381   \pgfpicture
3382   \pgfrememberpicturepositiononpagetrue
3383   \pgfcoordinate { \@@_env: - col - 2 }
3384   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3385   \str_if_empty:NF \l_@@_name_str
3386   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3387   \endpgfpicture
3388 }

```

We begin a loop over the columns. The integer `\g_tmpa_int` will be the number of the current column. This integer is used for the Tikz nodes.

```

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

```

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

```

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

```

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

```

3408 \pgfpicture
3409 \pgfrememberpicturepositiononpagetrue
3410 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3411 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3412 \str_if_empty:NF \l_@@_name_str
3413 {
3414   \pgfnodealias
3415   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3416   { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3417 }
3418 \endpgfpicture
3419 }

3420 &
3421 \omit

```

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

```

3422 \int_if_zero:nT \g_@@_col_total_int
3423 { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3424 \skip_horizontal:N \g_tmpa_skip
3425 \int_gincr:N \g_tmpa_int
3426 \bool_lazy_any:nF
3427 {
3428   \g_@@_delims_bool
3429   \l_@@_tabular_bool
3430   { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3431   \l_@@_exterior_arraycolsep_bool
3432   \l_@@_bar_at_end_of_pream_bool
3433 }
3434 { \skip_horizontal:N -\col@sep }
3435 \bool_if:NT \l_@@_code_before_bool
3436 {
3437   \hbox
3438   {
3439     \skip_horizontal:N -0.5\arrayrulewidth

```

With an environment `{Matrix}`, you want to remove the exterior `\arraycolsep` but we don't know the number of columns (since there is no preamble) and that's why we can't put `@{}` at the end of the preamble. That's why we remove a `\arraycolsep` now.

```

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

3468 \bool_if:NT \g_@@_last_col_found_bool
3469 {
3470   \hbox_overlap_right:n
3471   {
3472     \skip_horizontal:N \g_@@_width_last_col_dim
3473     \skip_horizontal:N \col@sep
3474     \bool_if:NT \l_@@_code_before_bool
3475     {
3476       \pgfsys@markposition
3477       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3478     }
3479     \pgfpicture

```

```

3480 \pgfrememberpicturepositiononpagetrue
3481 \pgfcoordinate
3482 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3483 \pgfpintorigin
3484 \str_if_empty:NF \l_@@_name_str
3485 {
3486   \pgfnodealias
3487   {
3488     \l_@@_name_str - col
3489     - \int_eval:n { \g_@@_col_total_int + 1 }
3490   }
3491   { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3492 }
3493 \endpgfpicture
3494 }
3495 }
3496 % \cr
3497 }

```

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

```

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

```

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

```

3502 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3503 \bool_gset_true:N \g_@@_after_col_zero_bool
3504 \@@_begin_of_row:

```

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

```

3505 \hbox_set:Nw \l_@@_cell_box
3506 \@@_math_toggle:
3507 \@@_tuning_key_small:

```

We insert `\l_@@_code_for_first_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

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

```

Be careful: despite this letter `l` the cells of the “first column” are composed in a `R` manner since they are composed in a `\hbox_overlap_left:n`.

```

3519 l
3520 <
3521 {
3522   \@@_math_toggle:
3523   \hbox_set_end:
3524   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3525   \@@_adjust_size_box:
3526   \@@_update_for_first_and_last_row:

```


We actualise the width of the “first column” because we will use this width after the construction of the array.

```

3527 \dim_gset:Nn \g_@@_width_first_col_dim
3528 { \dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box } }

```

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

```

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

```

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

```

3542 \tl_const:Nn \c_@@_preamble_last_col_tl
3543 {
3544   >
3545   {
3546     \bool_set_true:N \l_@@_in_last_col_bool

```

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

```

3547 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:

```

With the flag `\g_@@_last_col_found_bool`, we will know that the “last column” is really used.

```

3548 \bool_gset_true:N \g_@@_last_col_found_bool
3549 \int_gincr:N \c@jCol
3550 \int_gset_eq:NN \g_@@_col_total_int \c@jCol

```

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

```

3551 \hbox_set:Nw \l_@@_cell_box
3552 \@@_math_toggle:
3553 \@@_tuning_key_small:

```

We insert `\l_@@_code_for_last_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```

3554 \int_compare:nNnT \c@iRow > \c_zero_int
3555 {
3556   \bool_lazy_or:nnT
3557   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3558   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3559   {
3560     \l_@@_code_for_last_col_tl
3561     \xglobal \colorlet { nicematrix-last-col } { . }
3562   }
3563 }
3564 }
3565 1
3566 <
3567 {
3568   \@@_math_toggle:
3569   \hbox_set_end:
3570   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3571   \@@_adjust_size_box:
3572   \@@_update_for_first_and_last_row:

```

We actualise the width of the “last column” because we will use this width after the construction of the array.

```

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

```

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

```

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

```

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

```

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

```

We put `.` and `.` for the delimiters but, in fact, that doesn’t matter because these arguments won’t be used in `{NiceArrayWithDelims}` (because the flag `\g_@@_delims_bool` is set to false).

```

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

```

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

```

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

```

13 The environment `{NiceMatrix}` and its variants

```

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

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

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

```

We define also an environment `{NiceMatrix}`

```

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

```

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

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

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

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

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

```
3673   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3674     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3675   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3676   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3677   \tl_if_empty:NF \l_@@_short_caption_tl
3678   {
3679     \tl_if_empty:NT \l_@@_caption_tl
3680     {
3681       \@@_error_or_warning:n { short-caption-without-caption }
3682       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3683     }
3684   }
3685   \tl_if_empty:NF \l_@@_label_tl
3686   {
3687     \tl_if_empty:NT \l_@@_caption_tl
3688     { \@@_error_or_warning:n { label-without-caption } }
3689   }
3690   \NewDocumentEnvironment { TabularNote } { b }
3691   {
3692     \bool_if:NTF \l_@@_in_code_after_bool
3693       { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3694     {
3695       \tl_if_empty:NF \g_@@_tabularnote_tl
3696       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3697       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3698     }
3699   }
3700   { }
3701   \@@_settings_for_tabular:
3702   \NiceArray { #2 }
3703 }
3704 {
3705   \endNiceArray
3706   \bool_if:NT \c_@@_testphase_table_bool
3707     { \UseTaggingSocket { tbl / hmode / end } }
3708 }
3709 \cs_new_protected:Npn \@@_settings_for_tabular:
3710 {
3711   \bool_set_true:N \l_@@_tabular_bool
3712   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3713   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3714   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3715 }
```

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

```

3723 \NiceArray { #3 }
3724 }
3725 {
3726 \endNiceArray
3727 \int_if_zero:nT \g_@@_total_X_weight_int
3728 { \@@_error:n { NiceTabularX~without~X } }
3729 }

3730 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3731 {
3732 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3733 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3734 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3735 \@@_settings_for_tabular:
3736 \NiceArray { #3 }
3737 }
3738 { \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`).

```

3739 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3740 {
3741 \bool_lazy_all:nT
3742 {
3743 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3744 \l_@@_hvlines_bool
3745 { ! \g_@@_delims_bool }
3746 { ! \l_@@_except_borders_bool }
3747 }
3748 {
3749 \bool_set_true:N \l_@@_except_borders_bool
3750 \clist_if_empty:NF \l_@@_corners_clist
3751 { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3752 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3753 {
3754 \@@_stroke_block:nnn
3755 {
3756 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3757 draw = \l_@@_rules_color_tl
3758 }
3759 { 1-1 }
3760 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3761 }
3762 }
3763 }

3764 \cs_new_protected:Npn \@@_after_array:
3765 {

```

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`.

```

3766 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3767 \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.

```

3768     \bool_if:NT \g_@@_last_col_found_bool
3769     { \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`.

```

3770     \bool_if:NT \l_@@_last_col_without_value_bool
3771     { \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.

```

3772     \bool_if:NT \l_@@_last_row_without_value_bool
3773     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3774     \tl_gput_right:Ne \g_@@_aux_tl
3775     {
3776       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3777       {
3778         \int_use:N \l_@@_first_row_int ,
3779         \int_use:N \c@iRow ,
3780         \int_use:N \g_@@_row_total_int ,
3781         \int_use:N \l_@@_first_col_int ,
3782         \int_use:N \c@jCol ,
3783         \int_use:N \g_@@_col_total_int
3784       }
3785     }

```

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`).

```

3786     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3787     {
3788       \tl_gput_right:Ne \g_@@_aux_tl
3789       {
3790         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3791         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3792       }
3793     }
3794     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3795     {
3796       \tl_gput_right:Ne \g_@@_aux_tl
3797       {
3798         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3799         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3800         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3801         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3802       }
3803     }

```

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

```

3804     \@@_create_diag_nodes:

```

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

```

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

```

```

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

```

3834     \bool_if:NT \l_@@_parallelize_diags_bool
3835     {
3836         \int_gzero_new:N \g_@@_ddots_int
3837         \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.

```

3838         \dim_gzero_new:N \g_@@_delta_x_one_dim
3839         \dim_gzero_new:N \g_@@_delta_y_one_dim
3840         \dim_gzero_new:N \g_@@_delta_x_two_dim
3841         \dim_gzero_new:N \g_@@_delta_y_two_dim
3842     }
3843     \int_zero_new:N \l_@@_initial_i_int
3844     \int_zero_new:N \l_@@_initial_j_int
3845     \int_zero_new:N \l_@@_final_i_int
3846     \int_zero_new:N \l_@@_final_j_int
3847     \bool_set_false:N \l_@@_initial_open_bool
3848     \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.

```

3849     \bool_if:NT \l_@@_small_bool
3850     {
3851         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3852         \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.

```

3853         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3854         { 0.6 \l_@@_xdots_shorten_start_dim }
3855         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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

```

3856         { 0.6 \l_@@_xdots_shorten_end_dim }
3857     }

```

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

```

3858     \@@_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.

```

3859     \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-*}`).

```

3860     \@@_adjust_pos_of_blocks_seq:
3861     \@@_deal_with_rounded_corners:
3862     \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3863     \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3864     \IfPackageLoadedT { tikz }
3865     {
3866         \tikzset
3867         {
3868             every-picture / .style =
3869             {
3870                 overlay ,
3871                 remember~picture ,
3872                 name~prefix = \@@_env: -
3873             }
3874         }
3875     }
3876     \bool_if:NT \c_@@_tagging_array_bool
3877     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3878     \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3879     \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3880     \cs_set_eq:NN \OverBrace \@@_OverBrace
3881     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3882     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3883     \cs_set_eq:NN \line \@@_line
3884     \g_@@_pre_code_after_tl
3885     \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.

```

3886     \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).

```

3887     \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`).

```

3888     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3889     { \@@_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:`.

```

3890 \bool_set_true:N \l_@@_in_code_after_bool
3891 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3892 \scan_stop:
3893 \tl_gclear:N \g_nicematrix_code_after_tl
3894 \group_end:

```

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

```

3895 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3896 \tl_if_empty:NF \g_@@_pre_code_before_tl
3897 {
3898   \tl_gput_right:Ne \g_@@_aux_tl
3899   {
3900     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3901     { \exp_not:o \g_@@_pre_code_before_tl }
3902   }
3903   \tl_gclear:N \g_@@_pre_code_before_tl
3904 }
3905 \tl_if_empty:NF \g_nicematrix_code_before_tl
3906 {
3907   \tl_gput_right:Ne \g_@@_aux_tl
3908   {
3909     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3910     { \exp_not:o \g_nicematrix_code_before_tl }
3911   }
3912   \tl_gclear:N \g_nicematrix_code_before_tl
3913 }
3914 \str_gclear:N \g_@@_name_env_str
3915 \@@_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`.

```

3916 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3917 }

```

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`.

```

3918 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3919 { \keys_set:nn { nicematrix / CodeAfter } { #1 } }

```

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

```

3920 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3921 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3922 \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3923 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3924 }

```

The following command must *not* be protected.

```

3925 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3926 {
3927   { #1 }
3928   { #2 }
3929   {
3930     \int_compare:nNnTF { #3 } > { 99 }
3931     { \int_use:N \c@iRow }
3932     { #3 }
3933   }
3934   {
3935     \int_compare:nNnTF { #4 } > { 99 }
3936     { \int_use:N \c@jCol }
3937     { #4 }
3938   }
3939   { #5 }
3940 }

```

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).

```

3941 \hook_gput_code:nnn { begindocument } { . }
3942 {
3943   \cs_new_protected:Npe \@@_draw_dotted_lines:
3944   {
3945     \c_@@_pgfortikzpicture_tl
3946     \@@_draw_dotted_lines_i:
3947     \c_@@_endpgfortikzpicture_tl
3948   }
3949 }

```

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

```

3950 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3951 {
3952   \pgfrememberpicturepositiononpagetrue
3953   \pgfrelevantforpicturesizefalse
3954   \g_@@_HVdotsfor_lines_tl
3955   \g_@@_Vdots_lines_tl
3956   \g_@@_Ddots_lines_tl
3957   \g_@@_Iddots_lines_tl
3958   \g_@@_Cdots_lines_tl
3959   \g_@@_Ldots_lines_tl
3960 }

3961 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3962 {
3963   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3964   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3965 }

```

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

```

3966 \pgfdeclareshape { @@_diag_node }
3967 {
3968   \savedanchor { \five }
3969   {
3970     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

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

```

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).

```

3986 \cs_new_protected:Npn \@@_create_diag_nodes:
3987 {
3988   \pgfpicture
3989   \pgfrememberpicturepositiononpagetrue
3990   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3991   {
3992     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3993     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3994     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3995     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3996     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3997     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3998     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3999     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4000     \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.

```

4001     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4002     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4003     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4004     \str_if_empty:NF \l_@@_name_str
4005     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4006   }

```

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

```

4007     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4008     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4009     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4010     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4011     \pgfcoordinate
4012     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4013     \pgfnodealias
4014     { \@@_env: - last }
4015     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4016     \str_if_empty:NF \l_@@_name_str
4017     {
4018       \pgfnodealias
4019       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4020       { \@@_env: - \int_use:N \l_tmpa_int }
4021       \pgfnodealias
4022       { \l_@@_name_str - last }
4023       { \@@_env: - last }
4024     }

```

```

4025 \endpgfpicture
4026 }

```

16 We draw the dotted lines

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

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

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.

```

4027 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4028 {

```

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

```

4029 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4030 \int_set:Nn \l_@@_initial_i_int { #1 }
4031 \int_set:Nn \l_@@_initial_j_int { #2 }
4032 \int_set:Nn \l_@@_final_i_int { #1 }
4033 \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.

```

4034 \bool_set_false:N \l_@@_stop_loop_bool
4035 \bool_do_until:Nn \l_@@_stop_loop_bool
4036 {
4037   \int_add:Nn \l_@@_final_i_int { #3 }
4038   \int_add:Nn \l_@@_final_j_int { #4 }
4039   \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.

```

4040     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4041     \if_int_compare:w #3 = \c_one_int
4042     \bool_set_true:N \l_@@_final_open_bool
4043   \else:
4044     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4045     \bool_set_true:N \l_@@_final_open_bool
4046   \fi:
4047 \fi:
4048 \else:
4049   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4050   \if_int_compare:w #4 = -1
4051   \bool_set_true:N \l_@@_final_open_bool
4052   \fi:
4053 \else:
4054   \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4055   \if_int_compare:w #4 = \c_one_int
4056   \bool_set_true:N \l_@@_final_open_bool
4057   \fi:
4058 \fi:
4059 \fi:
4060 \fi:
4061 \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.

```

4062   {

```

We do a step backwards.

```

4063     \int_sub:Nn \l_@@_final_i_int { #3 }
4064     \int_sub:Nn \l_@@_final_j_int { #4 }
4065     \bool_set_true:N \l_@@_stop_loop_bool
4066   }

```

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`.

```

4067   {
4068     \cs_if_exist:cTF
4069     {
4070       @@ _ dotted _
4071       \int_use:N \l_@@_final_i_int -
4072       \int_use:N \l_@@_final_j_int
4073     }
4074     {
4075       \int_sub:Nn \l_@@_final_i_int { #3 }
4076       \int_sub:Nn \l_@@_final_j_int { #4 }
4077       \bool_set_true:N \l_@@_final_open_bool
4078       \bool_set_true:N \l_@@_stop_loop_bool
4079     }
4080     {
4081       \cs_if_exist:cTF
4082       {
4083         pgf @ sh @ ns @ \@@_env:
4084         - \int_use:N \l_@@_final_i_int
4085         - \int_use:N \l_@@_final_j_int
4086       }
4087       { \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.

```

4088   {

```

```

4089         \cs_set_nopar:cpn
4090         {
4091             @@ _ dotted _
4092             \int_use:N \l_@@_final_i_int -
4093             \int_use:N \l_@@_final_j_int
4094         }
4095         { }
4096     }
4097 }
4098 }
4099 }

```

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

```

4100     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4101     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4102     \bool_do_until:Nn \l_@@_stop_loop_bool
4103     {
4104         \int_sub:Nn \l_@@_initial_i_int { #3 }
4105         \int_sub:Nn \l_@@_initial_j_int { #4 }
4106         \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.

```

4107         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4108         \if_int_compare:w #3 = \c_one_int
4109             \bool_set_true:N \l_@@_initial_open_bool
4110         \else:

```

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

```

4111         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4112             \bool_set_true:N \l_@@_initial_open_bool
4113         \fi:
4114     \fi:
4115     \else:
4116         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4117             \if_int_compare:w #4 = \c_one_int
4118                 \bool_set_true:N \l_@@_initial_open_bool
4119             \fi:
4120         \else:
4121             \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4122                 \if_int_compare:w #4 = -1
4123                     \bool_set_true:N \l_@@_initial_open_bool
4124                 \fi:
4125             \fi:
4126         \fi:
4127     \fi:
4128     \bool_if:NTF \l_@@_initial_open_bool
4129     {
4130         \int_add:Nn \l_@@_initial_i_int { #3 }
4131         \int_add:Nn \l_@@_initial_j_int { #4 }
4132         \bool_set_true:N \l_@@_stop_loop_bool
4133     }
4134     {
4135         \cs_if_exist:cTF
4136         {
4137             @@ _ dotted _
4138             \int_use:N \l_@@_initial_i_int -
4139             \int_use:N \l_@@_initial_j_int
4140         }

```

```

4141     {
4142         \int_add:Nn \l_@@_initial_i_int { #3 }
4143         \int_add:Nn \l_@@_initial_j_int { #4 }
4144         \bool_set_true:N \l_@@_initial_open_bool
4145         \bool_set_true:N \l_@@_stop_loop_bool
4146     }
4147     {
4148         \cs_if_exist:cTF
4149         {
4150             pgf @ sh @ ns @ \@@_env:
4151             - \int_use:N \l_@@_initial_i_int
4152             - \int_use:N \l_@@_initial_j_int
4153         }
4154         { \bool_set_true:N \l_@@_stop_loop_bool }
4155         {
4156             \cs_set_nopar:cpn
4157             {
4158                 @@ _ dotted _
4159                 \int_use:N \l_@@_initial_i_int -
4160                 \int_use:N \l_@@_initial_j_int
4161             }
4162             { }
4163         }
4164     }
4165 }
4166 }

```

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.

```

4167 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4168 {
4169     { \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`.

```

4170     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4171     { \int_use:N \l_@@_final_i_int }
4172     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4173     { } % for the name of the block
4174 }
4175 }

```

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.

```

4176 \cs_new_protected:Npn \@@_open_shorten:
4177 {
4178     \bool_if:NT \l_@@_initial_open_bool
4179     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4180     \bool_if:NT \l_@@_final_open_bool
4181     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4182 }

```

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).

```

4183 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4184 {
4185     \int_set_eq:NN \l_@@_row_min_int \c_one_int

```

```

4186 \int_set_eq:NN \l_@@_col_min_int \c_one_int
4187 \int_set_eq:NN \l_@@_row_max_int \c@iRow
4188 \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`.

```

4189 \seq_if_empty:NF \g_@@_submatrix_seq
4190 {
4191   \seq_map_inline:Nn \g_@@_submatrix_seq
4192     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4193 }
4194 }

```

#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.

```

4195 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4196 {
4197   \if_int_compare:w #3 > #1
4198   \else:
4199     \if_int_compare:w #1 > #5
4200     \else:
4201       \if_int_compare:w #4 > #2
4202       \else:
4203         \if_int_compare:w #2 > #6
4204         \else:
4205           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4206           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4207           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4208           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4209         \fi:
4210       \fi:
4211     \fi:
4212   \fi:
4213 }

4214 \cs_new_protected:Npn \@@_set_initial_coords:
4215 {
4216   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4217   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4218 }
4219 \cs_new_protected:Npn \@@_set_final_coords:
4220 {

```



```

4221 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4222 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4223 }
4224 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4225 {
4226   \pgfpointanchor
4227   {
4228     \@@_env:
4229     - \int_use:N \l_@@_initial_i_int
4230     - \int_use:N \l_@@_initial_j_int
4231   }
4232   { #1 }
4233   \@@_set_initial_coords:
4234 }
4235 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4236 {
4237   \pgfpointanchor
4238   {
4239     \@@_env:
4240     - \int_use:N \l_@@_final_i_int
4241     - \int_use:N \l_@@_final_j_int
4242   }
4243   { #1 }
4244   \@@_set_final_coords:
4245 }
4246 \cs_new_protected:Npn \@@_open_x_initial_dim:
4247 {
4248   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4249   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4250   {
4251     \cs_if_exist:cT
4252     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4253     {
4254       \pgfpointanchor
4255       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4256       { west }
4257       \dim_set:Nn \l_@@_x_initial_dim
4258       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4259     }
4260   }

```

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

```

4261 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4262 {
4263   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4264   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4265   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4266 }
4267 }
4268 \cs_new_protected:Npn \@@_open_x_final_dim:
4269 {
4270   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4271   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4272   {
4273     \cs_if_exist:cT
4274     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4275     {
4276       \pgfpointanchor
4277       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4278       { east }
4279       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4280       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4281     }

```

```
4282 }
```

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

```
4283 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4284 {
4285   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4286   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4287   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4288 }
4289 }
```

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.

```
4290 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4291 {
4292   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4293   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4294   {
4295     \@@_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.

```
4296   \group_begin:
4297   \@@_open_shorten:
4298   \int_if_zero:nTF { #1 }
4299   { \color { nicematrix-first-row } }
4300   {
```

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.

```
4301     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4302     { \color { nicematrix-last-row } }
4303   }
4304   \keys_set:nn { nicematrix / xdots } { #3 }
4305   \@@_color:o \l_@@_xdots_color_tl
4306   \@@_actually_draw_Ldots:
4307   \group_end:
4308 }
4309 }
```

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`.

```
4310 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4311 {
4312   \bool_if:NTF \l_@@_initial_open_bool
4313   {
4314     \@@_open_x_initial_dim:
4315     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4316     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4317   }
4318   { \@@_set_initial_coords_from_anchor:n { base-east } }
```

```

4319 \bool_if:NTF \l_@@_final_open_bool
4320 {
4321   \@@_open_x_final_dim:
4322   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4323   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4324 }
4325 { \@@_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.

```

4326 \bool_lazy_all:nTF
4327 {
4328   \l_@@_initial_open_bool
4329   \l_@@_final_open_bool
4330   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4331 }
4332 {
4333   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4334   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4335 }

```

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 (?).

```

4336 {
4337   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4338   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4339 }
4340 \@@_draw_line:
4341 }

```

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.

```

4342 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4343 {
4344   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4345   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4346   {
4347     \@@_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.

```

4348 \group_begin:
4349   \@@_open_shorten:
4350   \int_if_zero:nTF { #1 }
4351   { \color { nicematrix-first-row } }
4352   {

```

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.

```

4353     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4354     { \color { nicematrix-last-row } }
4355   }
4356   \keys_set:nn { nicematrix / xdots } { #3 }
4357   \@@_color:o \l_@@_xdots_color_tl
4358   \@@_actually_draw_Cdots:
4359 \group_end:
4360 }
4361 }

```

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.

```

4362 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4363 {
4364   \bool_if:NTF \l_@@_initial_open_bool
4365     { \@@_open_x_initial_dim: }
4366     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4367   \bool_if:NTF \l_@@_final_open_bool
4368     { \@@_open_x_final_dim: }
4369     { \@@_set_final_coords_from_anchor:n { mid-west } }
4370   \bool_lazy_and:nnTF
4371     \l_@@_initial_open_bool
4372     \l_@@_final_open_bool
4373   {
4374     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4375     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4376     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4377     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4378     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4379   }
4380   {
4381     \bool_if:NT \l_@@_initial_open_bool
4382       { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4383     \bool_if:NT \l_@@_final_open_bool
4384       { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4385   }
4386   \@@_draw_line:
4387 }
4388 \cs_new_protected:Npn \@@_open_y_initial_dim:
4389 {
4390   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4391   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4392   {
4393     \cs_if_exist:cT
4394       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4395     {
4396       \pgfpointanchor
4397         { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4398         { north }
4399       \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4400         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4401     }
4402   }
4403   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4404   {
4405     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4406     \dim_set:Nn \l_@@_y_initial_dim
4407     {
4408       \fp_to_dim:n
4409       {
4410         \pgf@y
4411         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4412       }
4413     }
4414   }
4415 }

```

```

4416 \cs_new_protected:Npn \@@_open_y_final_dim:
4417 {
4418   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4419   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4420   {
4421     \cs_if_exist:cT
4422     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4423     {
4424       \pgfpointanchor
4425       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4426       { south }
4427       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4428       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4429     }
4430   }
4431   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4432   {
4433     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4434     \dim_set:Nn \l_@@_y_final_dim
4435     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4436   }
4437 }

```

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.

```

4438 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4439 {
4440   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4441   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4442   {
4443     \@@_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.

```

4444   \group_begin:
4445   \@@_open_shorten:
4446   \int_if_zero:nTF { #2 }
4447   { \color { nicematrix-first-col } }
4448   {
4449     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4450     { \color { nicematrix-last-col } }
4451   }
4452   \keys_set:nn { nicematrix / xdots } { #3 }
4453   \@@_color:o \l_@@_xdots_color_tl
4454   \@@_actually_draw_Vdots:
4455   \group_end:
4456 }
4457 }

```

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`.

```

4458 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4459 {

```

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

```
4460 \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.

```
4461 {
4462   \@@_open_y_initial_dim:
4463   \@@_open_y_final_dim:
4464   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4465 {
4466   \@@_qpoint:n { col - 1 }
4467   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4468   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4469   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4470   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4471 }
4472 {
4473   \bool_lazy_and:nnTF
4474   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4475   { \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”.

```
4476 {
4477   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4478   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4479   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4480   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4481   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4482 }
```

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

```
4483 {
4484   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4485   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4486   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4487   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4488 }
4489 }
4490 }
```

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.

```
4491 {
4492   \bool_set_false:N \l_tmpa_bool
4493   \bool_if:NF \l_@@_initial_open_bool
4494   {
4495     \bool_if:NF \l_@@_final_open_bool
4496     {
4497       \@@_set_initial_coords_from_anchor:n { south-west }
4498       \@@_set_final_coords_from_anchor:n { north-west }
4499       \bool_set:Nn \l_tmpa_bool
4500       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4501     }
4502   }
```

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

```
4503 \bool_if:NTF \l_@@_initial_open_bool
4504 {
4505   \@@_open_y_initial_dim:
4506   \@@_set_final_coords_from_anchor:n { north }
4507   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4508 }
4509 {
4510   \@@_set_initial_coords_from_anchor:n { south }
4511   \bool_if:NTF \l_@@_final_open_bool
```

```
4512         \@@_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.

```
4513     {
4514         \@@_set_final_coords_from_anchor:n { north }
4515         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4516         {
4517             \dim_set:Nn \l_@@_x_initial_dim
4518             {
4519                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4520                 \l_@@_x_initial_dim \l_@@_x_final_dim
4521             }
4522         }
4523     }
4524 }
4525 }
4526 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4527 \@@_draw_line:
4528 }
```

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.

```
4529 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4530 {
4531     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4532     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4533     {
4534         \@@_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.

```
4535     \group_begin:
4536     \@@_open_shorten:
4537     \keys_set:nn { nicematrix / xdots } { #3 }
4538     \@@_color:o \l_@@_xdots_color_tl
4539     \@@_actually_draw_Ddots:
4540     \group_end:
4541 }
4542 }
```

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.`

```
4543 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4544 {
4545     \bool_if:NTF \l_@@_initial_open_bool
4546     {
4547         \@@_open_y_initial_dim:
4548         \@@_open_x_initial_dim:
```

```

4549     }
4550     { \@@_set_initial_coords_from_anchor:n { south-east } }
4551     \bool_if:NTF \l_@@_final_open_bool
4552     {
4553         \@@_open_x_final_dim:
4554         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4555     }
4556     { \@@_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.

```

4557     \bool_if:NT \l_@@_parallelize_diags_bool
4558     {
4559         \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).

```

4560         \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.

```

4561         {
4562             \dim_gset:Nn \g_@@_delta_x_one_dim
4563             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4564             \dim_gset:Nn \g_@@_delta_y_one_dim
4565             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4566         }

```

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`.

```

4567         {
4568             \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4569             {
4570                 \dim_set:Nn \l_@@_y_final_dim
4571                 {
4572                     \l_@@_y_initial_dim +
4573                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4574                     \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4575                 }
4576             }
4577         }
4578     }
4579     \@@_draw_line:
4580 }

```

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.

```

4581 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4582 {
4583     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4584     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4585     {
4586         \@@_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.

```

4587     \group_begin:
4588     \@@_open_shorten:
4589     \keys_set:nn { nicematrix / xdots } { #3 }
4590     \@@_color:o \l_@@_xdots_color_tl
4591     \@@_actually_draw_Iddots:
4592     \group_end:
4593 }
4594 }

```


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.`

```

4595 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4596 {
4597   \bool_if:NTF \l_@@_initial_open_bool
4598   {
4599     \@@_open_y_initial_dim:
4600     \@@_open_x_initial_dim:
4601   }
4602   { \@@_set_initial_coords_from_anchor:n { south-west } }
4603   \bool_if:NTF \l_@@_final_open_bool
4604   {
4605     \@@_open_y_final_dim:
4606     \@@_open_x_final_dim:
4607   }
4608   { \@@_set_final_coords_from_anchor:n { north-east } }
4609   \bool_if:NT \l_@@_parallelize_diags_bool
4610   {
4611     \int_gincr:N \g_@@_iddots_int
4612     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4613     {
4614       \dim_gset:Nn \g_@@_delta_x_two_dim
4615       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4616       \dim_gset:Nn \g_@@_delta_y_two_dim
4617       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4618     }
4619     {
4620       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4621       {
4622         \dim_set:Nn \l_@@_y_final_dim
4623         {
4624           \l_@@_y_initial_dim +
4625           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4626           \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4627         }
4628       }
4629     }
4630   }
4631   \@@_draw_line:
4632 }

```

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

```

4633 \cs_new_protected:Npn \@@_draw_line:
4634 {
4635   \pgfrememberpicturerepositiononpagetrue
4636   \pgf@relevantforpicturesizefalse
4637   \bool_lazy_or:nnTF
4638     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4639     \l_@@_dotted_bool
4640     \@@_draw_standard_dotted_line:
4641     \@@_draw_unstandard_dotted_line:
4642 }

```

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.

```

4643 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4644 {
4645   \begin { scope }
4646     \@@_draw_unstandard_dotted_line:o
4647     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4648 }

```

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.

```

4649 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4650 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4651 {
4652   \@@_draw_unstandard_dotted_line:nooo
4653   { #1 }
4654   \l_@@_xdots_up_tl
4655   \l_@@_xdots_down_tl
4656   \l_@@_xdots_middle_tl
4657 }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4658 \hook_gput_code:nnn { begindocument } { . }
4659 {
4660   \IfPackageLoadedT { tikz }
4661   {
4662     \tikzset
4663     {
4664       @@_node_above / .style = { sloped , above } ,
4665       @@_node_below / .style = { sloped , below } ,
4666       @@_node_middle / .style =
4667       {
4668         sloped ,
4669         inner~sep = \c_@@_innersep_middle_dim
4670       }
4671     }
4672   }
4673 }

```

```

4674 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4675 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4676 {

```

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.

```

4677 \dim_zero_new:N \l_@@_l_dim
4678 \dim_set:Nn \l_@@_l_dim
4679 {
4680 \fp_to_dim:n
4681 {
4682 sqrt
4683 (
4684 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4685 +
4686 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4687 )
4688 }
4689 }

```

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.

```

4690 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4691 {
4692 \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4693 \@@_draw_unstandard_dotted_line_i:
4694 }

```

If the key `xdots/horizontal-labels` has been used.

```

4695 \bool_if:NT \l_@@_xdots_h_labels_bool
4696 {
4697 \tikzset
4698 {
4699 @@_node_above / .style = { auto = left } ,
4700 @@_node_below / .style = { auto = right } ,
4701 @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4702 }
4703 }
4704 \tl_if_empty:nF { #4 }
4705 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4706 \draw
4707 [ #1 ]
4708 ( \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).

```

4709 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4710 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4711 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4712 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4713 \end { scope }
4714 }
4715 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4716 {
4717 \dim_set:Nn \l_tmpa_dim
4718 {
4719 \l_@@_x_initial_dim
4720 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )

```

```

4721     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4722   }
4723   \dim_set:Nn \l_tmpb_dim
4724   {
4725     \l_@@_y_initial_dim
4726     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4727     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4728   }
4729   \dim_set:Nn \l_@@_tmpc_dim
4730   {
4731     \l_@@_x_final_dim
4732     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4733     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4734   }
4735   \dim_set:Nn \l_@@_tmpd_dim
4736   {
4737     \l_@@_y_final_dim
4738     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4739     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4740   }
4741   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4742   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4743   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4744   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4745 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4746 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4747 {
4748   \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.

```

4749     \dim_zero_new:N \l_@@_l_dim
4750     \dim_set:Nn \l_@@_l_dim
4751     {
4752       \fp_to_dim:n
4753       {
4754         sqrt
4755         (
4756           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4757           +
4758           ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4759         )
4760       }
4761     }

```

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.

```

4762     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4763     {
4764       \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4765       \@@_draw_standard_dotted_line_i:
4766     }
4767   \group_end:
4768   \bool_lazy_all:nF
4769   {
4770     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4771     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4772     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }

```

```

4773     }
4774     \l_@@_labels_standard_dotted_line:
4775 }
4776 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4777 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4778 {

```

The number of dots will be $\l_1\text{tmpa_int} + 1$.

```

4779     \int_set:Nn \l_1tmpa_int
4780     {
4781         \dim_ratio:nn
4782         {
4783             \l_@@_l_dim
4784             - \l_@@_xdots_shorten_start_dim
4785             - \l_@@_xdots_shorten_end_dim
4786         }
4787         \l_@@_xdots_inter_dim
4788     }

```

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.

```

4789     \dim_set:Nn \l_1tmpa_dim
4790     {
4791         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4792         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4793     }
4794     \dim_set:Nn \l_1tmpb_dim
4795     {
4796         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4797         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4798     }

```

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.

```

4799     \dim_gadd:Nn \l_@@_x_initial_dim
4800     {
4801         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4802         \dim_ratio:nn
4803         {
4804             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4805             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4806         }
4807         { 2 \l_@@_l_dim }
4808     }
4809     \dim_gadd:Nn \l_@@_y_initial_dim
4810     {
4811         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4812         \dim_ratio:nn
4813         {
4814             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_1tmpa_int
4815             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4816         }
4817         { 2 \l_@@_l_dim }
4818     }
4819     \pgf@relevantforpicturesizefalse
4820     \int_step_inline:nnn \c_zero_int \l_1tmpa_int
4821     {
4822         \pgfpathcircle
4823         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4824         { \l_@@_xdots_radius_dim }
4825         \dim_add:Nn \l_@@_x_initial_dim \l_1tmpa_dim
4826         \dim_add:Nn \l_@@_y_initial_dim \l_1tmpb_dim
4827     }

```

```

4828 \pgfusepathqfill
4829 }

4830 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4831 {
4832   \pgfscope
4833   \pgftransformshift
4834   {
4835     \pgfpointlineattime { 0.5 }
4836     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4837     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4838   }
4839   \fp_set:Nn \l_tmpa_fp
4840   {
4841     atand
4842     (
4843       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4844       \l_@@_x_final_dim - \l_@@_x_initial_dim
4845     )
4846   }
4847   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4848   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4849   \tl_if_empty:NF \l_@@_xdots_middle_tl
4850   {
4851     \begin { pgfscope }
4852     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4853     \pgfnode
4854     { rectangle }
4855     { center }
4856     {
4857       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4858       {
4859         \c_math_toggle_token
4860         \scriptstyle \l_@@_xdots_middle_tl
4861         \c_math_toggle_token
4862       }
4863     }
4864     { }
4865     {
4866       \pgfsetfillcolor { white }
4867       \pgfusepath { fill }
4868     }
4869     \end { pgfscope }
4870   }
4871   \tl_if_empty:NF \l_@@_xdots_up_tl
4872   {
4873     \pgfnode
4874     { rectangle }
4875     { south }
4876     {
4877       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4878       {
4879         \c_math_toggle_token
4880         \scriptstyle \l_@@_xdots_up_tl
4881         \c_math_toggle_token
4882       }
4883     }
4884     { }
4885     { \pgfusepath { } }
4886   }
4887   \tl_if_empty:NF \l_@@_xdots_down_tl
4888   {
4889     \pgfnode

```

```

4890     { rectangle }
4891     { north }
4892     {
4893       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4894       {
4895         \c_math_toggle_token
4896         \scriptstyle \l_@@_xdots_down_tl
4897         \c_math_toggle_token
4898       }
4899     }
4900     { }
4901     { \pgfusepath { } }
4902   }
4903 \endpgfscope
4904 }

```

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.

```

4905 \hook_gput_code:nnn { begindocument } { . }
4906 {
4907   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4908   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4909   \cs_new_protected:Npn \@@_Ldots
4910     { \@@_collect_options:n { \@@_Ldots_i } }
4911   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4912     {
4913       \int_if_zero:nTF \c@jCol
4914       { \@@_error:nn { in~first~col } \Ldots }
4915       {
4916         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4917         { \@@_error:nn { in~last~col } \Ldots }
4918         {
4919           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4920           { #1 , down = #2 , up = #3 , middle = #4 }
4921         }
4922       }
4923       \bool_if:NF \l_@@_nullify_dots_bool
4924       { \phantom { \ensuremath { \@@_old_ldots } } }
4925       \bool_gset_true:N \g_@@_empty_cell_bool
4926     }

4927   \cs_new_protected:Npn \@@_Cdots
4928     { \@@_collect_options:n { \@@_Cdots_i } }
4929   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4930     {
4931       \int_if_zero:nTF \c@jCol
4932       { \@@_error:nn { in~first~col } \Cdots }
4933       {
4934         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int

```

```

4935         { \@@_error:nn { in~last~col } \Cdots }
4936     {
4937         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4938         { #1 , down = #2 , up = #3 , middle = #4 }
4939     }
4940 }
4941 \bool_if:NF \l_@@_nullify_dots_bool
4942 { \phantom { \ensuremath { \@@_old_cdots } } }
4943 \bool_gset_true:N \g_@@_empty_cell_bool
4944 }

4945 \cs_new_protected:Npn \@@_Vdots
4946 { \@@_collect_options:n { \@@_Vdots_i } }
4947 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4948 {
4949     \int_if_zero:nTF \c@iRow
4950     { \@@_error:nn { in~first~row } \Vdots }
4951     {
4952         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4953         { \@@_error:nn { in~last~row } \Vdots }
4954         {
4955             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4956             { #1 , down = #2 , up = #3 , middle = #4 }
4957         }
4958     }
4959     \bool_if:NF \l_@@_nullify_dots_bool
4960     { \phantom { \ensuremath { \@@_old_vdots } } }
4961     \bool_gset_true:N \g_@@_empty_cell_bool
4962 }

4963 \cs_new_protected:Npn \@@_Ddots
4964 { \@@_collect_options:n { \@@_Ddots_i } }
4965 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4966 {
4967     \int_case:nnF \c@iRow
4968     {
4969         0 { \@@_error:nn { in~first~row } \Ddots }
4970         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4971     }
4972     {
4973         \int_case:nnF \c@jCol
4974         {
4975             0 { \@@_error:nn { in~first~col } \Ddots }
4976             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4977         }
4978         {
4979             \keys_set_known:nn { nicematrix / Ddots } { #1 }
4980             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4981             { #1 , down = #2 , up = #3 , middle = #4 }
4982         }
4983     }
4984 }
4985 \bool_if:NF \l_@@_nullify_dots_bool
4986 { \phantom { \ensuremath { \@@_old_ddots } } }
4987 \bool_gset_true:N \g_@@_empty_cell_bool
4988 }

4989 \cs_new_protected:Npn \@@_Iddots
4990 { \@@_collect_options:n { \@@_Iddots_i } }
4991 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4992 {

```



```

4993 \int_case:nnF \c@iRow
4994 {
4995     0 { \@@_error:nn { in~first~row } \Iddots }
4996     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4997 }
4998 {
4999     \int_case:nnF \c@jCol
5000     {
5001         0 { \@@_error:nn { in~first~col } \Iddots }
5002         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5003     }
5004     {
5005         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5006         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5007         { #1 , down = #2 , up = #3 , middle = #4 }
5008     }
5009 }
5010 \bool_if:NF \l_@@_nullify_dots_bool
5011 { \phantom { \ensuremath { \@@_old_iddots } } }
5012 \bool_gset_true:N \g_@@_empty_cell_bool
5013 }
5014 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5015 \keys_define:nn { nicematrix / Ddots }
5016 {
5017     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5018     draw-first .default:n = true ,
5019     draw-first .value_forbidden:n = true
5020 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5021 \cs_new_protected:Npn \@@_Hspace:
5022 {
5023     \bool_gset_true:N \g_@@_empty_cell_bool
5024     \hspace
5025 }

```

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`.

```

5026 \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`.

```

5027 \cs_new:Npn \@@_Hdotsfor:
5028 {
5029     \bool_lazy_and:nnTF
5030     { \int_if_zero_p:n \c@jCol }
5031     { \int_if_zero_p:n \l_@@_first_col_int }
5032     {
5033         \bool_if:NTF \g_@@_after_col_zero_bool
5034         {
5035             \multicolumn { 1 } { c } { }
5036             \@@_Hdotsfor_i
5037         }
5038         { \@@_fatal:n { Hdotsfor~in~col~0 } }
5039     }
5040 }

```

```

5041     \multicolumn { 1 } { c } { }
5042     \@@_Hdotsfor_i
5043   }
5044 }

```

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:`).

```

5045 \hook_gput_code:nnn { begindocument } { . }
5046 {
5047   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5048   \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.

```

5049   \cs_new_protected:Npn \@@_Hdotsfor_i
5050     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5051   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5052     {
5053       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5054       {
5055         \@@_Hdotsfor:nnnn
5056         { \int_use:N \c@iRow }
5057         { \int_use:N \c@jCol }
5058         { #2 }
5059         {
5060           #1 , #3 ,
5061           down = \exp_not:n { #4 } ,
5062           up = \exp_not:n { #5 } ,
5063           middle = \exp_not:n { #6 }
5064         }
5065       }
5066       \prg_replicate:nn { #2 - 1 }
5067       {
5068         &
5069         \multicolumn { 1 } { c } { }
5070         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5071       }
5072     }
5073 }

```

```

5074 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5075 {
5076   \bool_set_false:N \l_@@_initial_open_bool
5077   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5078   \int_set:Nn \l_@@_initial_i_int { #1 }
5079   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5080   \int_compare:nNnTF { #2 } = \c_one_int
5081   {
5082     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5083     \bool_set_true:N \l_@@_initial_open_bool
5084   }
5085   {
5086     \cs_if_exist:cTF
5087     {
5088       pgf @ sh @ ns @ \@@_env:
5089       - \int_use:N \l_@@_initial_i_int
5090       - \int_eval:n { #2 - 1 }
5091     }
5092     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5093   }

```

```

5094         \int_set:Nn \l_@@_initial_j_int { #2 }
5095         \bool_set_true:N \l_@@_initial_open_bool
5096     }
5097 }
5098 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5099 {
5100     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5101     \bool_set_true:N \l_@@_final_open_bool
5102 }
5103 {
5104     \cs_if_exist:cTF
5105     {
5106         pgf @ sh @ ns @ \@@_env:
5107         - \int_use:N \l_@@_final_i_int
5108         - \int_eval:n { #2 + #3 }
5109     }
5110     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5111     {
5112         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5113         \bool_set_true:N \l_@@_final_open_bool
5114     }
5115 }
5116 \group_begin:
5117 \@@_open_shorten:
5118 \int_if_zero:nTF { #1 }
5119 { \color { nicematrix-first-row } }
5120 {
5121     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5122     { \color { nicematrix-last-row } }
5123 }
5124
5125 \keys_set:nn { nicematrix / xdots } { #4 }
5126 \@@_color:o \l_@@_xdots_color_tl
5127 \@@_actually_draw_Ldots:
5128 \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).

```

5129     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5130     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5131 }

5132 \hook_gput_code:nnn { begindocument } { . }
5133 {
5134     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } { } } }
5135     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5136     \cs_new_protected:Npn \@@_Vdotsfor:
5137     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5138     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5139     {
5140         \bool_gset_true:N \g_@@_empty_cell_bool
5141         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5142         {
5143             \@@_Vdotsfor:nnnn
5144             { \int_use:N \c@iRow }
5145             { \int_use:N \c@jCol }
5146             { #2 }
5147             {
5148                 #1 , #3 ,
5149                 down = \exp_not:n { #4 } ,
5150                 up = \exp_not:n { #5 } ,

```

```

5151         middle = \exp_not:n { #6 }
5152     }
5153 }
5154 }
5155 }

```

```

5156 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5157 {
5158     \bool_set_false:N \l_@@_initial_open_bool
5159     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5160     \int_set:Nn \l_@@_initial_j_int { #2 }
5161     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5162     \int_compare:nNnTF { #1 } = \c_one_int
5163     {
5164         \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5165         \bool_set_true:N \l_@@_initial_open_bool
5166     }
5167     {
5168         \cs_if_exist:cTF
5169         {
5170             pgf @ sh @ ns @ \@@_env:
5171             - \int_eval:n { #1 - 1 }
5172             - \int_use:N \l_@@_initial_j_int
5173         }
5174         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5175         {
5176             \int_set:Nn \l_@@_initial_i_int { #1 }
5177             \bool_set_true:N \l_@@_initial_open_bool
5178         }
5179     }
5180     \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5181     {
5182         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5183         \bool_set_true:N \l_@@_final_open_bool
5184     }
5185     {
5186         \cs_if_exist:cTF
5187         {
5188             pgf @ sh @ ns @ \@@_env:
5189             - \int_eval:n { #1 + #3 }
5190             - \int_use:N \l_@@_final_j_int
5191         }
5192         { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5193         {
5194             \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5195             \bool_set_true:N \l_@@_final_open_bool
5196         }
5197     }
5198     \group_begin:
5199     \@@_open_shorten:
5200     \int_if_zero:nTF { #2 }
5201     { \color { nicematrix-first-col } }
5202     {
5203         \int_compare:nNnT { #2 } = \g_@@_col_total_int
5204         { \color { nicematrix-last-col } }
5205     }
5206     \keys_set:nn { nicematrix / xdots } { #4 }
5207     \@@_color:o \l_@@_xdots_color_tl
5208     \@@_actually_draw_Vdots:
5209     \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).

```

5210 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5211 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5212 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5213 \NewDocumentCommand \@@_rotate: { 0 { } }
5214 {
5215   \peek_remove_spaces:n
5216   {
5217     \bool_gset_true:N \g_@@_rotate_bool
5218     \keys_set:nn { nicematrix / rotate } { #1 }
5219   }
5220 }

5221 \keys_define:nn { nicematrix / rotate }
5222 {
5223   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5224   c .value_forbidden:n = true ,
5225   unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5226 }

```

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

```

5227 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5228 {
5229   \tl_if_empty:nTF { #2 }
5230   { #1 }
5231   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5232 }
5233 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5234 { \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).

```

5235 \hook_gput_code:nnn { begindocument } { . }
5236 {

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5237 \cs_set_nopar:Npn \l_@@_argspec_tl
5238 { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5239 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5240 \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5241 {
5242   \group_begin:
5243   \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5244   \@@_color:o \l_@@_xdots_color_tl
5245   \use:e
5246   {
5247     \@@_line_i:nn
5248     { \@@_double_int_eval:n #2 - \q_stop }
5249     { \@@_double_int_eval:n #3 - \q_stop }
5250   }
5251   \group_end:
5252 }
5253 }
5254 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5255 {
5256   \bool_set_false:N \l_@@_initial_open_bool
5257   \bool_set_false:N \l_@@_final_open_bool
5258   \bool_lazy_or:nnTF
5259   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5260   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5261   { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5262 { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5263 }
5264 \hook_gput_code:nnn { begindocument } { . }
5265 {
5266   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5267   {

```

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:.`

```

5268 \c_@@_pgfortikzpicture_tl
5269 \@@_draw_line_iii:nn { #1 } { #2 }
5270 \c_@@_endpgfortikzpicture_tl
5271 }
5272 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5273 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5274 {
5275   \pgfrememberpicturepositiononpagetrue
5276   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5277   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5278   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5279   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5280   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5281   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5282   \@@_draw_line:
5283 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

#1 is the first row *after* the scope of the instructions in #2

```
5284 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5285 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```
5286 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5287 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5288 {
5289   \tl_gput_right:Ne \g_@@_row_style_tl
5290   {
```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```
5291   \exp_not:N
5292   \@@_if_row_less_than:nn
5293   { \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`).

```
5294   { \exp_not:n { #1 } \scan_stop: }
5295 }
5296 }
```

```
5297 \keys_define:nn { nicematrix / RowStyle }
5298 {
5299   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5300   cell-space-top-limit .value_required:n = true ,
5301   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5302   cell-space-bottom-limit .value_required:n = true ,
5303   cell-space-limits .meta:n =
5304   {
5305     cell-space-top-limit = #1 ,
5306     cell-space-bottom-limit = #1 ,
5307   } ,
5308   color .tl_set:N = \l_@@_color_tl ,
5309   color .value_required:n = true ,
5310   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5311   bold .default:n = true ,
5312   nb-rows .code:n =
5313   { \str_if_eq:eeTF { #1 } { * }
5314     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5315     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5316   nb-rows .value_required:n = true ,
5317   rowcolor .tl_set:N = \l_tmpa_tl ,
5318   rowcolor .value_required:n = true ,
5319   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5320 }
```

```

5321 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5322 {
5323   \group_begin:
5324   \tl_clear:N \l_tmpa_tl
5325   \tl_clear:N \l_@@_color_tl
5326   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5327   \dim_zero:N \l_tmpa_dim
5328   \dim_zero:N \l_tmpb_dim
5329   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5330   \tl_if_empty:NF \l_tmpa_tl
5331   {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

5332     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5333     {

```

The command \@@_exp_color_arg:No is *fully expandable*.

```

5334         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5335         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5336         { \int_use:N \c@iRow - * }
5337     }

```

Then, the other rows (if there is several rows).

```

5338     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5339     {
5340       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5341       {
5342         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5343         {
5344           \int_eval:n { \c@iRow + 1 }
5345           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5346         }
5347       }
5348     }
5349   }
5350   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5351   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5352   {
5353     \@@_put_in_row_style:e
5354     {
5355       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5356       {

```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```

5357         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5358         { \dim_use:N \l_tmpa_dim }
5359       }
5360     }
5361   }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5362   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5363   {
5364     \@@_put_in_row_style:e
5365     {
5366       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5367       {
5368         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5369         { \dim_use:N \l_tmpb_dim }
5370       }
5371     }
5372   }

```


`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5373   \tl_if_empty:NF \l_@@_color_tl
5374   {
5375     \@@_put_in_row_style:e
5376     {
5377       \mode_leave_vertical:
5378       \@@_color:n { \l_@@_color_tl }
5379     }
5380   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5381   \bool_if:NT \l_@@_bold_row_style_bool
5382   {
5383     \@@_put_in_row_style:n
5384     {
5385       \exp_not:n
5386       {
5387         \if_mode_math:
5388           \c_math_toggle_token
5389           \bfseries \boldmath
5390           \c_math_toggle_token
5391         \else:
5392           \bfseries \boldmath
5393         \fi:
5394       }
5395     }
5396   }
5397   \group_end:
5398   \g_@@_row_style_tl
5399   \ignorespaces
5400 }

```

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).

```

5401 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5402 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5403 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5404 {

```

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.

```
5405 \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`.

```
5406 \str_if_in:nnF { #1 } { !! }
5407 {
5408   \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5409   { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5410 }
5411 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5412 {
5413   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5414   \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5415 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5416   { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5417 }
```

The following command must be used within a `\pgfpicture`.

```
5418 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5419 {
5420   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5421   {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5422   \group_begin:
5423   \pgfsetcornersarced
5424   {
5425     \pgfpoint
5426     { \l_@@_tab_rounded_corners_dim }
5427     { \l_@@_tab_rounded_corners_dim }
5428   }
```

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.

```
5429   \bool_if:NTF \l_@@_hvlines_bool
5430   {
5431     \pgfpathrectanglecorners
5432     {
5433       \pgfpointadd
5434       { \@@_qpoint:n { row-1 } }
5435       { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5436     }
5437     {
5438       \pgfpointadd
5439       {
5440         \@@_qpoint:n
5441         { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5442       }
5443       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5444     }
5445   }
5446   {
```

```

5447         \pgfpathrectanglecorners
5448         { \@@_qpoint:n { row-1 } }
5449         {
5450             \pgfpointadd
5451             {
5452                 \@@_qpoint:n
5453                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5454             }
5455             { \pgfpoint \c_zero_dim \arrayrulewidth }
5456         }
5457     }
5458     \pgfusepath { clip }
5459     \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5460     }
5461 }

```

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).

```

5462 \cs_new_protected:Npn \@@_actually_color:
5463 {
5464     \pgfpicture
5465     \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.

```

5466     \@@_clip_with_rounded_corners:
5467     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5468     {
5469         \int_compare:nNnTF { ##1 } = \c_one_int
5470         {
5471             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5472             \use:c { g_@@_color _ 1 _tl }
5473             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5474         }
5475         {
5476             \begin { pgfscope }
5477                 \@@_color_opacity ##2
5478                 \use:c { g_@@_color _ ##1 _tl }
5479                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5480                 \pgfusepath { fill }
5481             \end { pgfscope }
5482         }
5483     }
5484     \endpgfpicture
5485 }

```

The following command will extract the potential key opacity in its optional argument (between square brackets) and (of course) then apply the command \color.

```

5486 \cs_new_protected:Npn \@@_color_opacity
5487 {
5488     \peek_meaning:NTF [
5489         { \@@_color_opacity:w }
5490         { \@@_color_opacity:w [ ] }
5491     }

```

The command \@@_color_opacity:w takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5492 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5493 {
5494     \tl_clear:N \l_tmpa_tl
5495     \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.

```

5496     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5497     \tl_if_empty:NTF \l_tmpb_tl
5498       { \@declaredcolor }
5499       { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5500   }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5501 \keys_define:nn { nicematrix / color-opacity }
5502 {
5503   opacity .tl_set:N          = \l_tmpa_tl ,
5504   opacity .value_required:n = true
5505 }

5506 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5507 {
5508   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5509   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5510   \@@_cartesian_path:
5511 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5512 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5513 {
5514   \tl_if_blank:nF { #2 }
5515   {
5516     \@@_add_to_colors_seq:en
5517     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5518     { \@@_cartesian_color:nn { #3 } { - } }
5519   }
5520 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5521 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5522 {
5523   \tl_if_blank:nF { #2 }
5524   {
5525     \@@_add_to_colors_seq:en
5526     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5527     { \@@_cartesian_color:nn { - } { #3 } }
5528   }
5529 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5530 \NewDocumentCommand \@@_rectanglecolor { 0 { } m 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     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5537   }
5538 }

```

The last argument is the radius of the corners of the rectangle.

```

5539 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m 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     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5546     }
5547 }

```

The last argument is the radius of the corners of the rectangle.

```

5548 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5549 {
5550   \@@_cut_on_hyphen:w #1 \q_stop
5551   \tl_clear_new:N \l_@@_tmpc_tl
5552   \tl_clear_new:N \l_@@_tmpd_tl
5553   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5554   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5555   \@@_cut_on_hyphen:w #2 \q_stop
5556   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5557   \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`.

```

5558   \@@_cartesian_path:n { #3 }
5559 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5560 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5561 {
5562   \clist_map_inline:nn { #3 }
5563   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5564 }

5565 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5566 {
5567   \int_step_inline:nn \c@iRow
5568   {
5569     \int_step_inline:nn \c@jCol
5570     {
5571       \int_if_even:nTF { #####1 + ##1 }
5572       { \@@_cellcolor [ #1 ] { #2 } }
5573       { \@@_cellcolor [ #1 ] { #3 } }
5574       { ##1 - #####1 }
5575     }
5576   }
5577 }

```

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”.

```

5578 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5579 {
5580   \@@_rectanglecolor [ #1 ] { #2 }
5581   { 1 - 1 }
5582   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5583 }

5584 \keys_define:nn { nicematrix / rowcolors }
5585 {
5586   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5587   respect-blocks .default:n = true ,
5588   cols .tl_set:N = \l_@@_cols_tl ,

```

```

5589     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5590     restart .default:n = true ,
5591     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5592 }

```

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*.

```

5593 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5594 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5595     \group_begin:
5596     \seq_clear_new:N \l_@@_colors_seq
5597     \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5598     \tl_clear_new:N \l_@@_cols_tl
5599     \cs_set_nopar:Npn \l_@@_cols_tl { - }
5600     \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).

```

5601     \int_zero_new:N \l_@@_color_int
5602     \int_set_eq:NN \l_@@_color_int \c_one_int
5603     \bool_if:NT \l_@@_respect_blocks_bool
5604     {

```

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`).

```

5605         \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5606         \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5607         { \@@_not_in_exterior_p:nnnnn ##1 }
5608     }
5609     \pgfpicture
5610     \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5611     \clist_map_inline:nn { #2 }
5612     {
5613         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5614         \tl_if_in:NnTF \l_tmpa_tl { - }
5615         { \@@_cut_on_hyphen:w ##1 \q_stop }
5616         { \tl_set:Nn \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.

```

5617         \int_set:Nn \l_tmpa_int \l_tmpa_tl
5618         \int_set:Nn \l_@@_color_int
5619         { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5620         \int_zero_new:N \l_@@_tmpc_int
5621         \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5622         \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5623         {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5624         \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).

```

5625         \bool_if:NT \l_@@_respect_blocks_bool
5626         {
5627             \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5628             { \@@_intersect_our_row_p:nnnnn #####1 }
5629             \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5630     }
5631     \tl_set:No \l_@@_rows_tl
5632     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
\l_@@_tmpc_tl will be the color that we will use.
5633     \tl_clear_new:N \l_@@_color_tl
5634     \tl_set:Ne \l_@@_color_tl
5635     {
5636         \@@_color_index:n
5637         {
5638             \int_mod:nn
5639             { \l_@@_color_int - 1 }
5640             { \seq_count:N \l_@@_colors_seq }
5641             + 1
5642         }
5643     }
5644     \tl_if_empty:NF \l_@@_color_tl
5645     {
5646         \@@_add_to_colors_seq:ee
5647         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5648         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5649     }
5650     \int_incr:N \l_@@_color_int
5651     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5652 }
5653 }
5654 \endpgfpicture
5655 \group_end:
5656 }

```

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.

```

5657 \cs_new:Npn \@@_color_index:n #1
5658 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5659     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5660     { \@@_color_index:n { #1 - 1 } }
5661     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5662 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5663 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5664 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around `#3` and `#4` are mandatory.

```

5665 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5666 {
5667     \int_compare:nNnT { #3 } > \l_tmpb_int
5668     { \int_set:Nn \l_tmpb_int { #3 } }
5669 }

```

```

5670 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5671 {
5672     \int_if_zero:nTF { #4 }
5673     \prg_return_false:
5674     {
5675         \int_compare:nNnTF { #2 } > \c@jCol

```

```

5676         \prg_return_false:
5677         \prg_return_true:
5678     }
5679 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5680 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5681 {
5682     \int_compare:nNnTF { #1 } > \l_tmpa_int
5683     \prg_return_false:
5684     {
5685         \int_compare:nNnTF \l_tmpa_int > { #3 }
5686         \prg_return_false:
5687         \prg_return_true:
5688     }
5689 }

```

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).

```

5690 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5691 {
5692     \dim_compare:nNnTF { #1 } = \c_zero_dim
5693     {
5694         \bool_if:NTF
5695         \l_@@_nocolor_used_bool
5696         \@@_cartesian_path_normal_ii:
5697         {
5698             \clist_if_empty:NTF \l_@@_corners_cells_clist
5699             { \@@_cartesian_path_normal_i:n { #1 } }
5700             \@@_cartesian_path_normal_ii:
5701         }
5702     }
5703     { \@@_cartesian_path_normal_i:n { #1 } }
5704 }

```

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.

```

5705 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5706 {
5707     \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5708 \clist_map_inline:Nn \l_@@_cols_tl
5709 {
5710     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5711     \tl_if_in:NnTF \l_tmpa_tl { - }
5712     { \@@_cut_on_hyphen:w ##1 \q_stop }
5713     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5714     \tl_if_empty:NTF \l_tmpa_tl
5715     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5716     {
5717         \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5718         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5719     }
5720     \tl_if_empty:NTF \l_tmpb_tl
5721     { \tl_set:Nn \l_tmpb_tl { \int_use:N \c@jCol } }
5722     {

```



```

5723         \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5724         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5725     }
5726     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5727     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5728     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5729     \@@_qpoint:n { col - \l_tmpa_tl }
5730     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5731     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5732     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5733     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5734     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5735     \clist_map_inline:Nn \l_@@_rows_tl
5736     {
5737         \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5738         \tl_if_in:NnTF \l_tmpa_tl { - }
5739         { \@@_cut_on_hyphen:w ####1 \q_stop }
5740         { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5741         \tl_if_empty:NnTF \l_tmpa_tl
5742         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5743         {
5744             \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5745             { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5746         }
5747         \tl_if_empty:NnTF \l_tmpb_tl
5748         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5749         {
5750             \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5751             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5752         }
5753         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5754         { \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`.

```

5755         \cs_if_exist:cF
5756         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5757         {
5758             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5759             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5760             \@@_qpoint:n { row - \l_tmpa_tl }
5761             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5762             \pgfpathrectanglecorners
5763             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5764             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5765         }
5766     }
5767 }
5768 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5769 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5770 {
5771     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5772     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5773     \clist_map_inline:Nn \l_@@_cols_tl
5774     {
5775         \@@_qpoint:n { col - ##1 }
5776         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }

```

```

5777         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5778         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5779     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5780     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5781     \clist_map_inline:Nn \l_@@_rows_tl
5782     {
5783         \@@_if_in_corner:nF { #####1 - ##1 }
5784         {
5785             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5786             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5787             \@@_qpoint:n { row - #####1 }
5788             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5789             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5790             {
5791                 \pgfpathrectanglecorners
5792                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5793                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5794             }
5795         }
5796     }
5797 }
5798 }

```

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`).

```

5799 \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

```

5800 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5801 {
5802     \bool_set_true:N \l_@@_nocolor_used_bool
5803     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5804     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5805     \clist_map_inline:Nn \l_@@_rows_tl
5806     {
5807         \clist_map_inline:Nn \l_@@_cols_tl
5808         { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5809     }
5810 }

```

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.

```

5811 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5812 {
5813     \clist_set_eq:NN \l_tmpa_clist #1
5814     \clist_clear:N #1
5815     \clist_map_inline:Nn \l_tmpa_clist
5816     {
5817         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5818         \tl_if_in:NnTF \l_tmpa_tl { - }
5819         { \@@_cut_on_hyphen:w ##1 \q_stop }
5820         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5821         \bool_lazy_or:nnT
5822         { \str_if_eq_p:ee \l_tmpa_tl { * } }
5823         { \tl_if_blank_p:o \l_tmpa_tl }

```

```

5824     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5825 \bool_lazy_or:nnT
5826   { \str_if_eq_p:ee \l_tmpb_tl { * } }
5827   { \tl_if_blank_p:o \l_tmpb_tl }
5828   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5829 \int_compare:nNnT \l_tmpb_tl > #2
5830   { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5831 \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5832   { \clist_put_right:Nn #1 { ###1 } }
5833 }
5834 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5835 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5836 {
5837   \@@_test_color_inside:
5838   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5839   {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on latex and pdf_latex).

```

5840     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5841     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5842   }
5843   \ignorespaces
5844 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5845 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5846 {
5847   \@@_test_color_inside:
5848   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5849   {
5850     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5851     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5852     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5853   }
5854   \ignorespaces
5855 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5856 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5857 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5858 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5859 {
5860   \@@_test_color_inside:
5861   \peek_remove_spaces:n
5862   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5863 }

5864 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5865 {

```

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`.

```

5866 \seq_gclear:N \g_tmpa_seq
5867 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5868 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5869 \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`.

```

5870 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5871 {
5872   { \int_use:N \c@iRow }
5873   { \exp_not:n { #1 } }
5874   { \exp_not:n { #2 } }
5875   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5876 }
5877 }

```

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`).

```

5878 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5879 {
5880   \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).

```

5881   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5882   {
5883     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5884     {
5885       \@@_rowlistcolors
5886       [ \exp_not:n { #2 } ]
5887       { #1 - \int_eval:n { \c@iRow - 1 } }
5888       { \exp_not:n { #3 } }
5889       [ \exp_not:n { #4 } ]
5890     }
5891   }
5892 }

```

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.

```

5893 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5894 {
5895   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5896   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5897   \seq_gclear:N \g_@@_rowlistcolors_seq
5898 }

```

```

5899 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5900 {
5901   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5902   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5903 }

```

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.

```

5904 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5905 {

```

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).

```

5906   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5907   {

```

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).

```

5908     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5909     {
5910       \exp_not:N \columncolor [ #1 ]
5911       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5912     }
5913   }
5914 }

```

```

5915 \hook_gput_code:nnn { begindocument } { . }
5916 {
5917   \IfPackageLoadedTF { colortbl }
5918   {
5919     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5920     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5921     \cs_new_protected:Npn \@@_revert_colortbl:
5922     {
5923       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5924       {
5925         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5926         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5927       }
5928     }
5929   }
5930   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5931 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
5932 \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`.

```
5933 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5934 {
5935   \int_if_zero:nTF \l_@@_first_col_int
5936   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5937   {
5938     \int_if_zero:nTF \c@jCol
5939     {
5940       \int_compare:nNnF \c@iRow = { -1 }
5941       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5942     }
5943     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5944   }
5945 }
```

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.

```
5946 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5947 {
5948   \int_if_zero:nF \c@iRow
5949   {
5950     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5951     {
5952       \int_compare:nNnT \c@jCol > \c_zero_int
5953       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5954     }
5955   }
5956 }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5957 \keys_define:nn { nicematrix / Rules }
5958 {
5959   position .int_set:N = \l_@@_position_int ,
5960   position .value_required:n = true ,
5961   start .int_set:N = \l_@@_start_int ,
5962   end .code:n =
5963     \bool_lazy_or:nnTF
5964     { \tl_if_empty_p:n { #1 } }
5965     { \str_if_eq_p:ee { #1 } { last } }
5966     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5967     { \int_set:Nn \l_@@_end_int { #1 } }
5968 }
```

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.

```

5969 \keys_define:nn { nicematrix / RulesBis }
5970 {
5971   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5972   multiplicity .initial:n = 1 ,
5973   dotted .bool_set:N = \l_@@_dotted_bool ,
5974   dotted .initial:n = false ,
5975   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]`.

```

5976   color .code:n =
5977     \@@_set_CT@arc@:n { #1 }
5978     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5979   color .value_required:n = true ,
5980   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5981   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.

```

5982   tikz .code:n =
5983     \IfPackageLoadedTF { tikz }
5984       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5985       { \@@_error:n { tikz~without~tikz } } ,
5986   tikz .value_required:n = true ,
5987   total-width .dim_set:N = \l_@@_rule_width_dim ,
5988   total-width .value_required:n = true ,
5989   width .meta:n = { total-width = #1 } ,
5990   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5991 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5992 \cs_new_protected:Npn \@@_vline:n #1
5993 {

```

The group is for the options.

```

5994   \group_begin:
5995   \int_set_eq:NN \l_@@_end_int \c@iRow
5996   \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).

```

5997   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5998     \@@_vline_i:
5999   \group_end:
6000 }
6001 \cs_new_protected:Npn \@@_vline_i:
6002 {

```

`\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`.

```

6003   \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6004   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6005     \l_tmpa_tl
6006   {

```

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.

```

6007      \bool_gset_true:N \g_tmpa_bool
6008      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6009        { \@@_test_vline_in_block:nnnnn ##1 }
6010      \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6011        { \@@_test_vline_in_block:nnnnn ##1 }
6012      \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6013        { \@@_test_vline_in_stroken_block:nnnn ##1 }
6014      \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6015      \bool_if:NTF \g_tmpa_bool
6016        {
6017          \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.

```

6018          { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6019        }
6020      {
6021        \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6022        {
6023          \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6024          \@@_vline_ii:
6025          \int_zero:N \l_@@_local_start_int
6026        }
6027      }
6028    }
6029    \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6030    {
6031      \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6032      \@@_vline_ii:
6033    }
6034  }

6035  \cs_new_protected:Npn \@@_test_in_corner_v:
6036  {
6037    \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6038    {
6039      \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6040      { \bool_set_false:N \g_tmpa_bool }
6041    }
6042    {
6043      \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6044      {
6045        \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6046        { \bool_set_false:N \g_tmpa_bool }
6047        {
6048          \@@_if_in_corner:nT
6049          { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6050          { \bool_set_false:N \g_tmpa_bool }
6051        }
6052      }
6053    }
6054  }

6055  \cs_new_protected:Npn \@@_vline_ii:
6056  {
6057    \tl_clear:N \l_@@_tikz_rule_tl
6058    \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```



```

6059 \bool_if:NTF \l_@@_dotted_bool
6060 \@@_vline_iv:
6061 {
6062     \tl_if_empty:NTF \l_@@_tikz_rule_tl
6063     \@@_vline_iii:
6064     \@@_vline_v:
6065 }
6066 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6067 \cs_new_protected:Npn \@@_vline_iii:
6068 {
6069     \pgfpicture
6070     \pgfrememberpicturepositiononpagetrue
6071     \pgf@relevantforpicturesizefalse
6072     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6073     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6074     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6075     \dim_set:Nn \l_tmpb_dim
6076     {
6077         \pgf@x
6078         - 0.5 \l_@@_rule_width_dim
6079         +
6080         ( \arrayrulewidth * \l_@@_multiplicity_int
6081           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6082     }
6083     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6084     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6085     \bool_lazy_all:nT
6086     {
6087         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6088         { \cs_if_exist_p:N \CT@drsc@ }
6089         { ! \tl_if_blank_p:o \CT@drsc@ }
6090     }
6091     {
6092         \group_begin:
6093         \CT@drsc@
6094         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6095         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6096         \dim_set:Nn \l_@@_tmpd_dim
6097         {
6098             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6099             * ( \l_@@_multiplicity_int - 1 )
6100         }
6101         \pgfpathrectanglecorners
6102         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6103         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6104         \pgfusepath { fill }
6105         \group_end:
6106     }
6107     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6108     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6109     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6110     {
6111         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6112         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6113         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6114         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6115     }
6116     \CT@arc@
6117     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6118     \pgfsetrectcap
6119     \pgfusepathqstroke

```

```

6120 \endpgfpicture
6121 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6122 \cs_new_protected:Npn \@@_vline_iv:
6123 {
6124   \pgfpicture
6125   \pgfrememberpicturepositiononpagetrue
6126   \pgf@relevantforpicturesizefalse
6127   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6128   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6129   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6130   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6131   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6132   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6133   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6134   \CT@arc@
6135   \@@_draw_line:
6136   \endpgfpicture
6137 }

```

The following code is for the case when the user uses the key `tikz`.

```

6138 \cs_new_protected:Npn \@@_vline_v:
6139 {
6140   \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).

```

6141   \CT@arc@
6142   \tl_if_empty:NF \l_@@_rule_color_tl
6143   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6144   \pgfrememberpicturepositiononpagetrue
6145   \pgf@relevantforpicturesizefalse
6146   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6147   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6148   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6149   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6150   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6151   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6152   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6153   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6154   ( \l_tmpb_dim , \l_tmpa_dim ) --
6155   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6156   \end {tikzpicture }
6157 }

```

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).

```

6158 \cs_new_protected:Npn \@@_draw_vlines:
6159 {
6160   \int_step_inline:nnn
6161   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6162   {
6163     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6164     \c@jCol
6165     { \int_eval:n { \c@jCol + 1 } }
6166   }
6167   {
6168     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6169     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6170     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6171     }
6172 }

```

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}`.

```

6173 \cs_new_protected:Npn \@@_hline:n #1
6174 {

```

The group is for the options.

```

6175     \group_begin:
6176     \int_zero_new:N \l_@@_end_int
6177     \int_set_eq:NN \l_@@_end_int \c@jCol
6178     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6179     \@@_hline_i:
6180     \group_end:
6181 }
6182 \cs_new_protected:Npn \@@_hline_i:
6183 {
6184     \int_zero_new:N \l_@@_local_start_int
6185     \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`.

```

6186     \tl_set:Nn \l_tmpa_tl { \int_use:N \l_@@_position_int }
6187     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6188         \l_tmpb_tl
6189     {

```

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.

```

6190         \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6191         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6192             { \@@_test_hline_in_block:nnnnn ##1 }
6193         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6194             { \@@_test_hline_in_block:nnnnn ##1 }
6195         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6196             { \@@_test_hline_in_stroken_block:nnnn ##1 }
6197         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6198         \bool_if:NTF \g_tmpa_bool
6199             {
6200                 \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.

```

6201             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6202         }
6203     {
6204         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6205         {
6206             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6207             \@@_hline_ii:
6208             \int_zero:N \l_@@_local_start_int
6209         }
6210     }
6211 }
6212 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6213     {
6214         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6215         \@@_hline_ii:
6216     }
6217 }

6218 \cs_new_protected:Npn \@@_test_in_corner_h:
6219 {
6220     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6221     {
6222         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6223         { \bool_set_false:N \g_tmpa_bool }
6224     }
6225     {
6226         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6227         {
6228             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6229             { \bool_set_false:N \g_tmpa_bool }
6230             {
6231                 \@@_if_in_corner:nT
6232                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6233                 { \bool_set_false:N \g_tmpa_bool }
6234             }
6235         }
6236     }
6237 }

6238 \cs_new_protected:Npn \@@_hline_ii:
6239 {
6240     \tl_clear:N \l_@@_tikz_rule_tl
6241     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6242     \bool_if:NTF \l_@@_dotted_bool
6243     \@@_hline_iv:
6244     {
6245         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6246         \@@_hline_iii:
6247         \@@_hline_v:
6248     }
6249 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6250 \cs_new_protected:Npn \@@_hline_iii:
6251 {
6252     \pgfpicture
6253     \pgfrememberpicturepositiononpagetrue
6254     \pgf@relevantforpicturesizefalse
6255     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6256     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6257     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6258     \dim_set:Nn \l_tmpb_dim
6259     {
6260         \pgf@y
6261         - 0.5 \l_@@_rule_width_dim
6262         +
6263         ( \arrayrulewidth * \l_@@_multiplicity_int
6264           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6265     }
6266     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6267     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6268     \bool_lazy_all:nT
6269     {

```

```

6270 { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6271 { \cs_if_exist_p:N \CT@drsc@ }
6272 { ! \tl_if_blank_p:o \CT@drsc@ }
6273 }
6274 {
6275 \group_begin:
6276 \CT@drsc@
6277 \dim_set:Nn \l_@@_tmpd_dim
6278 {
6279 \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6280 * ( \l_@@_multiplicity_int - 1 )
6281 }
6282 \pgfpathrectanglecorners
6283 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6284 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6285 \pgfusepathqfill
6286 \group_end:
6287 }
6288 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6289 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6290 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6291 {
6292 \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6293 \dim_sub:Nn \l_tmpb_dim \doublerulesep
6294 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6295 \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6296 }
6297 \CT@arc@
6298 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6299 \pgfsetrectcap
6300 \pgfusepathqstroke
6301 \endpgfpicture
6302 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6303 \cs_new_protected:Npn \@@_hline_iv:
6304 {
6305 \pgfpicture
6306 \pgfrememberpicturepositiononpagetrue
6307 \pgf@relevantforpicturesizefalse
6308 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6309 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6310 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6311 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6312 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```

```

6313 \int_compare:nNtT \l_@@_local_start_int = \c_one_int
6314 {
6315     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6316     \bool_if:NF \g_@@_delims_bool
6317     { \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 $0.5 \text{ } \l_@@_xdots_inter_dim$ is *ad hoc* for a better result.

```

6318     \tl_if_eq:NnF \g_@@_left_delim_tl (
6319         { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6320     )
6321     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6322     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6323     \int_compare:nNtT \l_@@_local_end_int = \c@jCol
6324     {
6325         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6326         \bool_if:NF \g_@@_delims_bool
6327         { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6328         \tl_if_eq:NnF \g_@@_right_delim_tl )
6329         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6330     }
6331     \CT@arc@
6332     \@@_draw_line:
6333     \endpgfpicture
6334 }

```

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`).

```

6335 \cs_new_protected:Npn \@@_hline_v:
6336 {
6337     \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).

```

6338     \CT@arc@
6339     \tl_if_empty:NF \l_@@_rule_color_tl
6340     { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6341     \pgfrememberpicturepositiononpagetrue
6342     \pgf@relevantforpicturesizefalse
6343     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6344     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6345     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6346     \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6347     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6348     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6349     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6350     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6351     ( \l_tmpa_dim , \l_tmpb_dim ) --
6352     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6353     \end { tikzpicture }
6354 }

```

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).

```

6355 \cs_new_protected:Npn \@@_draw_hlines:
6356 {
6357     \int_step_inline:nnn
6358     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6359     {
6360         \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6361         \c@iRow
6362         { \int_eval:n { \c@iRow + 1 } }
6363     }
6364     {
6365         \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
6366         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6367         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6368     }
6369 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6370 \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.

```

6371 \cs_set:Npn \@@_Hline_i:n #1
6372 {
6373     \peek_remove_spaces:n
6374     {
6375         \peek_meaning:NTF \Hline
6376         { \@@_Hline_ii:nn { #1 + 1 } }
6377         { \@@_Hline_iii:n { #1 } }
6378     }
6379 }
6380 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6381 \cs_set:Npn \@@_Hline_iii:n #1
6382 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6383 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6384 {
6385     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6386     \skip_vertical:N \l_@@_rule_width_dim
6387     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6388     {
6389         \@@_hline:n
6390         {
6391             multiplicity = #1 ,
6392             position = \int_eval:n { \c@iRow + 1 } ,
6393             total-width = \dim_use:N \l_@@_rule_width_dim ,
6394             #2
6395         }
6396     }
6397     \egroup
6398 }

```

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`).

```

6399 \cs_new_protected:Npn \@@_custom_line:n #1
6400 {
6401     \str_clear_new:N \l_@@_command_str
6402     \str_clear_new:N \l_@@_ccommand_str
6403     \str_clear_new:N \l_@@_letter_str
6404     \tl_clear_new:N \l_@@_other_keys_tl
6405     \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6406 \bool_lazy_all:nTF
6407 {
6408   { \str_if_empty_p:N \l_@@_letter_str }
6409   { \str_if_empty_p:N \l_@@_command_str }
6410   { \str_if_empty_p:N \l_@@_ccommand_str }
6411 }
6412 { \@@_error:n { No~letter~and~no~command } }
6413 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6414 }
6415 \keys_define:nn { nicematrix / custom-line }
6416 {
6417   letter .str_set:N = \l_@@_letter_str ,
6418   letter .value_required:n = true ,
6419   command .str_set:N = \l_@@_command_str ,
6420   command .value_required:n = true ,
6421   ccommand .str_set:N = \l_@@_ccommand_str ,
6422   ccommand .value_required:n = true ,
6423 }
6424 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6425 \cs_new_protected:Npn \@@_custom_line_i:n #1
6426 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6427 \bool_set_false:N \l_@@_tikz_rule_bool
6428 \bool_set_false:N \l_@@_dotted_rule_bool
6429 \bool_set_false:N \l_@@_color_bool
6430 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6431 \bool_if:NT \l_@@_tikz_rule_bool
6432 {
6433   \IfPackageLoadedF { tikz }
6434   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6435   \bool_if:NT \l_@@_color_bool
6436   { \@@_error:n { color~in~custom~line~with~tikz } }
6437 }
6438 \bool_if:NT \l_@@_dotted_rule_bool
6439 {
6440   \int_compare:nNtT \l_@@_multiplicity_int > \c_one_int
6441   { \@@_error:n { key~multiplicity~with~dotted } }
6442 }
6443 \str_if_empty:NF \l_@@_letter_str
6444 {
6445   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6446   { \@@_error:n { Several~letters } }
6447   {
6448     \tl_if_in:NoTF
6449     \c_@@_forbidden_letters_str
6450     \l_@@_letter_str
6451     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6452   }

```

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.

```

6453 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6454 { \@@_v_custom_line:n { #1 } }
6455 }
6456 }
6457 }
6458 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6459 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6460 }

```



```

6461 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6462 \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.

```

6463 \keys_define:nn { nicematrix / custom-line-bis }
6464 {
6465   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6466   multiplicity .initial:n = 1 ,
6467   multiplicity .value_required:n = true ,
6468   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6469   color .value_required:n = true ,
6470   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6471   tikz .value_required:n = true ,
6472   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6473   dotted .value_forbidden:n = true ,
6474   total-width .code:n = { } ,
6475   total-width .value_required:n = true ,
6476   width .code:n = { } ,
6477   width .value_required:n = true ,
6478   sep-color .code:n = { } ,
6479   sep-color .value_required:n = true ,
6480   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6481 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6482 \bool_new:N \l_@@_dotted_rule_bool
6483 \bool_new:N \l_@@_tikz_rule_bool
6484 \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`.

```

6485 \keys_define:nn { nicematrix / custom-line-width }
6486 {
6487   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6488   multiplicity .initial:n = 1 ,
6489   multiplicity .value_required:n = true ,
6490   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6491   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6492   \bool_set_true:N \l_@@_total_width_bool ,
6493   total-width .value_required:n = true ,
6494   width .meta:n = { total-width = #1 } ,
6495   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6496 }

```

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`).

```

6497 \cs_new_protected:Npn \@@_h_custom_line:n #1
6498 {

```

We use `\cs_set:cpn` and not `\cs_new:cpn` because we want a local definition. Moreover, the command must *not* be protected since it begins with `\noalign` (which is in `\Hline`).

```

6499   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6500   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6501 }

```

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`).

```
6502 \cs_new_protected:Npn \@@_c_custom_line:n #1
6503 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6504   \exp_args:Nc \NewExpandableDocumentCommand
6505     { nicematrix - \l_@@_ccommand_str }
6506     { 0 { } m }
6507     {
6508       \noalign
6509       {
6510         \@@_compute_rule_width:n { #1 , ##1 }
6511         \skip_vertical:n { \l_@@_rule_width_dim }
6512         \clist_map_inline:nn
6513           { ##2 }
6514           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6515       }
6516     }
6517   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6518 }
```

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*.

```
6519 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6520 {
6521   \tl_if_in:nnTF { #2 } { - }
6522     { \@@_cut_on_hyphen:w #2 \q_stop }
6523     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6524   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6525     {
6526       \@@_hline:n
6527       {
6528         #1 ,
6529         start = \l_tmpa_tl ,
6530         end = \l_tmpb_tl ,
6531         position = \int_eval:n { \c@iRow + 1 } ,
6532         total-width = \dim_use:N \l_@@_rule_width_dim
6533       }
6534     }
6535 }

6536 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6537 {
6538   \bool_set_false:N \l_@@_tikz_rule_bool
6539   \bool_set_false:N \l_@@_total_width_bool
6540   \bool_set_false:N \l_@@_dotted_rule_bool
6541   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6542   \bool_if:NF \l_@@_total_width_bool
6543     {
6544       \bool_if:NTF \l_@@_dotted_rule_bool
6545         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6546         {
6547           \bool_if:NF \l_@@_tikz_rule_bool
6548             {
6549               \dim_set:Nn \l_@@_rule_width_dim
6550                 {
6551                   \arrayrulewidth * \l_@@_multiplicity_int
6552                   + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6553                 }
6554             }
6555         }
6556     }
6557 }
```

```

6558 \cs_new_protected:Npn \@@_v_custom_line:n #1
6559 {
6560   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6561   \tl_gput_right:Ne \g_@@_array_preamble_tl
6562     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6563   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6564     {
6565     \@@_vline:n
6566     {
6567       #1 ,
6568       position = \int_eval:n { \c@jCol + 1 } ,
6569       total-width = \dim_use:N \l_@@_rule_width_dim
6570     }
6571   }
6572   \@@_rec_preamble:n
6573 }

6574 \@@_custom_line:n
6575 { 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.

```

6576 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6577 {
6578   \int_compare:nNnT \l_tmpa_tl > { #1 }
6579   {
6580     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6581     {
6582       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6583       {
6584         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6585         { \bool_gset_false:N \g_tmpa_bool }
6586       }
6587     }
6588   }
6589 }

```

The same for vertical rules.

```

6590 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6591 {
6592   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6593   {
6594     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6595     {
6596       \int_compare:nNnT \l_tmpb_tl > { #2 }
6597       {
6598         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6599         { \bool_gset_false:N \g_tmpa_bool }
6600       }
6601     }
6602   }
6603 }

6604 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6605 {
6606   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6607   {
6608     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6609     {

```

```

6610         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6611         { \bool_gset_false:N \g_tmpa_bool }
6612         {
6613             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6614             { \bool_gset_false:N \g_tmpa_bool }
6615         }
6616     }
6617 }
6618 }

6619 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6620 {
6621     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6622     {
6623         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6624         {
6625             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6626             { \bool_gset_false:N \g_tmpa_bool }
6627             {
6628                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6629                 { \bool_gset_false:N \g_tmpa_bool }
6630             }
6631         }
6632     }
6633 }

```

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.

```

6634 \cs_new_protected:Npn \@@_compute_corners:
6635 {
6636     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6637     { \@@_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`).

```

6638     \clist_clear:N \l_@@_corners_cells_clist
6639     \clist_map_inline:Nn \l_@@_corners_cells_clist
6640     {
6641         \str_case:nnF { ##1 }
6642         {
6643             { NW }
6644             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6645             { NE }
6646             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6647             { SW }
6648             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6649             { SE }
6650             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6651         }
6652         { \@@_error:nn { bad~corner } { ##1 } }
6653     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6654     \clist_if_empty:NF \l_@@_corners_cells_clist
6655     {

```

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.

```

6656     \tl_gput_right:Np \g_@@_aux_tl
6657     {
6658         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6659         { \l_@@_corners_cells_clist }
6660     }
6661 }
6662 }

6663 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6664 {
6665     \int_step_inline:nnn { #1 } { #3 }
6666     {
6667         \int_step_inline:nnn { #2 } { #4 }
6668         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6669     }
6670 }

6671 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6672 {
6673     \cs_if_exist:cTF
6674     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6675     \prg_return_true:
6676     \prg_return_false:
6677 }

```

“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.

```

6678 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6679 {

```

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.

```

6680     \bool_set_false:N \l_tmpa_bool
6681     \int_zero_new:N \l_@@_last_empty_row_int
6682     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6683     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6684     {
6685         \bool_lazy_or:nnTF
6686         {
6687             \cs_if_exist_p:c
6688             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6689         }
6690         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6691         { \bool_set_true:N \l_tmpa_bool }
6692     }

```

```

6693         \bool_if:NF \l_tmpa_bool
6694         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6695     }
6696 }

```

Now, you determine the last empty cell in the row of number 1.

```

6697 \bool_set_false:N \l_tmpa_bool
6698 \int_zero_new:N \l_@@_last_empty_column_int
6699 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6700 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6701 {
6702     \bool_lazy_or:nnTF
6703     {
6704         \cs_if_exist_p:c
6705         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6706     }
6707     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6708     { \bool_set_true:N \l_tmpa_bool }
6709     {
6710         \bool_if:NF \l_tmpa_bool
6711         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6712     }
6713 }

```

Now, we loop over the rows.

```

6714 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6715 {

```

We treat the row number ##1 with another loop.

```

6716     \bool_set_false:N \l_tmpa_bool
6717     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6718     {
6719         \bool_lazy_or:nnTF
6720         { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6721         { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6722         { \bool_set_true:N \l_tmpa_bool }
6723         {
6724             \bool_if:NF \l_tmpa_bool
6725             {
6726                 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6727                 \clist_put_right:Nn
6728                 \l_@@_corners_cells_clist
6729                 { ##1 - #####1 }
6730                 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6731             }
6732         }
6733     }
6734 }
6735 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6736 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6737 \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.

```
6738 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6739 \keys_define:nn { nicematrix / NiceMatrixBlock }
6740 {
6741   auto-columns-width .code:n =
6742   {
6743     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6744     \dim_gzero_new:N \g_@@_max_cell_width_dim
6745     \bool_set_true:N \l_@@_auto_columns_width_bool
6746   }
6747 }

6748 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6749 {
6750   \int_gincr:N \g_@@_NiceMatrixBlock_int
6751   \dim_zero:N \l_@@_columns_width_dim
6752   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6753   \bool_if:NT \l_@@_block_auto_columns_width_bool
6754   {
6755     \cs_if_exist:cT
6756     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6757     {
6758       \dim_set:Nn \l_@@_columns_width_dim
6759       {
6760         \use:c
6761         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6762       }
6763     }
6764   }
6765 }
```

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`).

```
6766 {
6767   \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.

```
6768   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6769   {
6770     \bool_if:NT \l_@@_block_auto_columns_width_bool
6771     {
6772       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6773       \iow_shipout:Ne \@mainaux
6774       {
6775         \cs_gset:cpn
6776         { @@ _ 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.

```
6777         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6778       }
6779       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6780     }
6781   }
6782   \ignorespacesafterend
6783 }
```

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).

```

6784 \cs_new_protected:Npn \@@_create_extra_nodes:
6785 {
6786   \bool_if:nTF \l_@@_medium_nodes_bool
6787   {
6788     \bool_if:nTF \l_@@_large_nodes_bool
6789     \@@_create_medium_and_large_nodes:
6790     \@@_create_medium_nodes:
6791   }
6792   { \bool_if:nT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6793 }

```

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`.

```

6794 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6795 {
6796   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6797   {
6798     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6799     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6800     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6801     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6802   }
6803   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6804   {
6805     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6806     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6807     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6808     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6809   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6810   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6811   {
6812     \int_step_variable:nnNn
6813     \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.

```

6814         {
6815             \cs_if_exist:cT
6816             { \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`.

```

6817         {
6818             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6819             \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6820             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6821             \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6822             {
6823                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6824                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6825             }

```

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`.

```

6826             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6827             \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6828             { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6829             \seq_if_in:Nef \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6830             {
6831                 \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6832                 { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6833             }
6834         }
6835     }
6836 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6837 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6838 {
6839     \dim_compare:nNnT
6840     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6841     {
6842         \@@_qpoint:n { row - \@@_i: - base }
6843         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6844         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6845     }
6846 }
6847 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6848 {
6849     \dim_compare:nNnT
6850     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6851     {
6852         \@@_qpoint:n { col - \@@_j: }
6853         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6854         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6855     }
6856 }
6857 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6858 \cs_new_protected:Npn \@@_create_medium_nodes:
6859 {
6860     \pgfpicture
6861     \pgfrememberpicturepositiononpagetrue
6862     \pgf@relevantforpicturesizefalse
6863     \@@_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”.

```

6864     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6865     \@@_create_nodes:
6866     \endpgfpicture
6867 }

```

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:`.

```

6868 \cs_new_protected:Npn \@@_create_large_nodes:
6869 {
6870   \pgfpicture
6871   \pgfrememberpicturepositiononpagetrue
6872   \pgf@relevantforpicturesizefalse
6873   \@@_computations_for_medium_nodes:
6874   \@@_computations_for_large_nodes:
6875   \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6876   \@@_create_nodes:
6877   \endpgfpicture
6878 }
6879 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6880 {
6881   \pgfpicture
6882   \pgfrememberpicturepositiononpagetrue
6883   \pgf@relevantforpicturesizefalse
6884   \@@_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”.

```

6885     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6886     \@@_create_nodes:
6887     \@@_computations_for_large_nodes:
6888     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6889     \@@_create_nodes:
6890     \endpgfpicture
6891 }

```

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.

```

6892 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6893 {
6894   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6895   \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`.

```

6896   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6897   {
6898     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6899     {
6900       (
6901         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6902         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6903       )
6904       / 2
6905     }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6906     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6907     { l_@@_row_ \@@_i: _ min_dim }
6908   }
6909   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6910   {
6911     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6912     {
6913       (
6914         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6915         \dim_use:c
6916         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6917       )
6918       / 2
6919     }
6920     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6921     { l_@@_column _ \@@_j: _ max _ dim }
6922   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6923   \dim_sub:cn
6924   { l_@@_column _ 1 _ min _ dim }
6925   \l_@@_left_margin_dim
6926   \dim_add:cn
6927   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6928   \l_@@_right_margin_dim
6929 }

```

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).

```

6930 \cs_new_protected:Npn \@@_create_nodes:
6931 {
6932   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6933   {
6934     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6935     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6936     \@@_pgf_rect_node:nnnnn
6937     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6938     { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6939     { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6940     { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6941     { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6942     \str_if_empty:NF \l_@@_name_str
6943     {
6944       \pgfnodealias
6945       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6946       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6947     }
6948   }
6949 }

```

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 .

```

6950   \seq_map_pairwise_function:NNN
6951   \g_@@_multicolumn_cells_seq
6952   \g_@@_multicolumn_sizes_seq
6953   \@@_node_for_multicolumn:nn
6954 }

```

```

6955 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6956 {
6957   \cs_set_nopar:Npn \@@_i: { #1 }
6958   \cs_set_nopar:Npn \@@_j: { #2 }
6959 }

```

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”).

```

6960 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6961 {
6962   \@@_extract_coords_values: #1 \q_stop
6963   \@@_pgf_rect_node:nnnnn
6964   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6965   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6966   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6967   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6968   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6969   \str_if_empty:NF \l_@@_name_str
6970   {
6971     \pgfnodealias
6972     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6973     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6974   }
6975 }

```

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).

```

6976 \keys_define:nn { nicematrix / Block / FirstPass }
6977 {
6978   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
6979   \bool_set_true:N \l_@@_p_block_bool ,
6980   j .value_forbidden:n = true ,
6981   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6982   l .value_forbidden:n = true ,
6983   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6984   r .value_forbidden:n = true ,
6985   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6986   c .value_forbidden:n = true ,
6987   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6988   L .value_forbidden:n = true ,
6989   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6990   R .value_forbidden:n = true ,
6991   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6992   C .value_forbidden:n = true ,
6993   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6994   t .value_forbidden:n = true ,
6995   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6996   T .value_forbidden:n = true ,
6997   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6998   b .value_forbidden:n = true ,
6999   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7000   B .value_forbidden:n = true ,

```

```

7001 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7002 m .value_forbidden:n = true ,
7003 v-center .meta:n = m ,
7004 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7005 p .value_forbidden:n = true ,
7006 color .code:n =
7007   \@@_color:n { #1 }
7008   \tl_set_rescan:Nnn
7009     \l_@@_draw_tl
7010     { \char_set_catcode_other:N ! }
7011     { #1 } ,
7012 color .value_required:n = true ,
7013 respect-arraystretch .code:n =
7014   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7015 respect-arraystretch .value_forbidden:n = true ,
7016 }

```

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.

```

7017 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

7018 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7019 {

```

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).

```

7020   \peek_remove_spaces:n
7021   {
7022     \tl_if_blank:nTF { #2 }
7023     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7024     {
7025       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7026       \@@_Block_i_czech \@@_Block_i
7027       #2 \q_stop
7028     }
7029     { #1 } { #3 } { #4 }
7030   }
7031 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7032 \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`.

```

7033 {
7034   \char_set_catcode_active:N -
7035   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7036 }

```

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.

```

7037 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7038 {

```

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).

```

7039 \bool_lazy_or:nnTF
7040 { \tl_if_blank_p:n { #1 } }
7041 { \str_if_eq_p:ee { * } { #1 } }
7042 { \int_set:Nn \l_tmpa_int { 100 } }
7043 { \int_set:Nn \l_tmpa_int { #1 } }
7044 \bool_lazy_or:nnTF
7045 { \tl_if_blank_p:n { #2 } }
7046 { \str_if_eq_p:ee { * } { #2 } }
7047 { \int_set:Nn \l_tmpb_int { 100 } }
7048 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7049 \int_compare:nnTF \l_tmpb_int = \c_one_int
7050 {
7051   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7052   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7053   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7054 }
7055 { \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.

```

7056 \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7057 \tl_set:Ne \l_tmpa_tl
7058 {
7059   { \int_use:N \c@iRow }
7060   { \int_use:N \c@jCol }
7061   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7062   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7063 }

```

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).

```

7064 \bool_set_false:N \l_tmpa_bool
7065 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```

7066 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7067 \bool_case:nF
7068 {
7069   \l_tmpa_bool { \@@_Block_vii:eennn }
7070   \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.

```

7071 \l_@@_X_bool { \@@_Block_v:eennn }
7072 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7073 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7074 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7075 }
7076 { \@@_Block_v:eennn }
7077 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7078 }

```

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.

```

7079 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7080 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7081 {
7082   \int_gincr:N \g_@@_block_box_int
7083   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7084   {
7085     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7086     {
7087       \@@_actually_diagbox:nnnnnn
7088       { \int_use:N \c@iRow }
7089       { \int_use:N \c@jCol }
7090       { \int_eval:n { \c@iRow + #1 - 1 } }
7091       { \int_eval:n { \c@jCol + #2 - 1 } }
7092       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7093       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7094     }
7095   }
7096   \box_gclear_new:c
7097   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7098   \hbox_gset:cn
7099   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7100   {

```

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}`).

```

7101   \tl_if_empty:NTF \l_@@_color_tl
7102   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7103   { \@@_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`.

```

7104   \int_compare:nNnT { #1 } = \c_one_int
7105   {
7106     \int_if_zero:nTF \c@iRow
7107     {

```

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}$

```

```

7108         \cs_set_eq:NN \Block \@@_NullBlock:
7109         \l_@@_code_for_first_row_tl
7110     }
7111     {
7112         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7113         {
7114             \cs_set_eq:NN \Block \@@_NullBlock:
7115             \l_@@_code_for_last_row_tl
7116         }
7117     }
7118     \g_@@_row_style_tl
7119 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7120     \@@_reset_arraystretch:
7121     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7122     #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.).

```

7123     \@@_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}`.

```

7124     \bool_if:NTF \l_@@_tabular_bool
7125     {
7126         \bool_lazy_all:nTF
7127         {
7128             { \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.

```

7129         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7130         { ! \g_@@_rotate_bool }
7131     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7132     {
7133         \use:e
7134         {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7135         \exp_not:N \begin { minipage }%
7136         [ \str_lowercase:o \l_@@_vpos_block_str ]
7137         { \l_@@_col_width_dim }
7138         \str_case:on \l_@@_hpos_block_str
7139         { c \centering r \raggedleft l \raggedright }
7140     }

```



```

7141         #5
7142         \end { minipage }
7143     }

```

In the other cases, we use a `{tabular}`.

```

7144     {
7145         \use:e
7146         {
7147             \exp_not:N \begin { tabular }%
7148             [ \str_lowercase:o \l_@@_vpos_block_str ]
7149             { @ { } \l_@@_hpos_block_str @ { } }
7150         }
7151         #5
7152         \end { tabular }
7153     }
7154 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7155     {
7156         \c_math_toggle_token
7157         \use:e
7158         {
7159             \exp_not:N \begin { array }%
7160             [ \str_lowercase:o \l_@@_vpos_block_str ]
7161             { @ { } \l_@@_hpos_block_str @ { } }
7162         }
7163         #5
7164         \end { array }
7165         \c_math_toggle_token
7166     }
7167 }

```

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).

```

7168     \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.

```

7169     \int_compare:nNnT { #2 } = \c_one_int
7170     {
7171         \dim_gset:Nn \g_@@_blocks_wd_dim
7172         {
7173             \dim_max:nn
7174             \g_@@_blocks_wd_dim
7175             {
7176                 \box_wd:c
7177                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7178             }
7179         }
7180     }

```

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 explictely an option of vertical position.

```

7181     \bool_lazy_and:nnT
7182     { \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.

```

7183     { \str_if_empty_p:N \l_@@_vpos_block_str }
7184     {
7185         \dim_gset:Nn \g_@@_blocks_ht_dim

```

```

7186     {
7187         \dim_max:nn
7188         \g_@@_blocks_ht_dim
7189         {
7190             \box_ht:c
7191             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7192         }
7193     }
7194     \dim_gset:Nn \g_@@_blocks_dp_dim
7195     {
7196         \dim_max:nn
7197         \g_@@_blocks_dp_dim
7198         {
7199             \box_dp:c
7200             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7201         }
7202     }
7203 }
7204 \seq_gput_right:Ne \g_@@_blocks_seq
7205 {
7206     \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.

```

7207     {
7208         \exp_not:n { #3 } ,
7209         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7210     \bool_if:NT \g_@@_rotate_bool
7211     {
7212         \bool_if:NTF \g_@@_rotate_c_bool
7213         { m }
7214         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7215     }
7216 }
7217 {
7218     \box_use_drop:c
7219     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7220 }
7221 }
7222 \bool_set_false:N \g_@@_rotate_c_bool
7223 }

```

```

7224 \cs_new:Npn \@@_adjust_hpos_rotate:
7225 {
7226     \bool_if:NT \g_@@_rotate_bool
7227     {
7228         \str_set:Ne \l_@@_hpos_block_str
7229         {
7230             \bool_if:NTF \g_@@_rotate_c_bool
7231             { c }
7232             {
7233                 \str_case:onF \l_@@_vpos_block_str
7234                 { b l B l t r T r }
7235                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7236             }
7237         }
7238     }
7239 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7240 \cs_new_protected:Npn \@@_rotate_box_of_block:
7241 {
7242   \box_grotate:cn
7243   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7244   { 90 }
7245   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7246   {
7247     \vbox_gset_top:cn
7248     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7249     {
7250       \skip_vertical:n { 0.8 ex }
7251       \box_use:c
7252       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7253     }
7254   }
7255   \bool_if:NT \g_@@_rotate_c_bool
7256   {
7257     \hbox_gset:cn
7258     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7259     {
7260       \c_math_toggle_token
7261       \vcenter
7262       {
7263         \box_use:c
7264         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7265       }
7266       \c_math_toggle_token
7267     }
7268   }
7269 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. \@@_draw_blocks: and above all \@@_Block_v:nnnnn).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of key=values pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7270 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7271 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7272 {
7273   \seq_gput_right:Ne \g_@@_blocks_seq
7274   {
7275     \l_tmpa_tl
7276     { \exp_not:n { #3 } }
7277     {
7278       \bool_if:NTF \l_@@_tabular_bool
7279       {
7280         \group_begin:

```

The following command will be no-op when respect-arraystretch is in force.

```

7281 \@@_reset_arraystretch:
7282 \exp_not:n
7283 {
7284   \dim_zero:N \extrarowheight
7285   #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).

```

7286         \bool_if:NT \c_@@_testphase_table_bool
7287         { \tag_stop:n { table } }
7288         \use:e
7289         {
7290             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7291             { @ { } \l_@@_hpos_block_str @ { } }
7292         }
7293         #5
7294         \end { tabular }
7295     }
7296     \group_end:
7297 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7298     {
7299         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7300         \@@_reset_arraystretch:
7301         \exp_not:n
7302         {
7303             \dim_zero:N \extrarowheight
7304             #4
7305             \c_math_toggle_token
7306             \use:e
7307             {
7308                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7309                 { @ { } \l_@@_hpos_block_str @ { } }
7310             }
7311             #5
7312             \end { array }
7313             \c_math_toggle_token
7314         }
7315         \group_end:
7316     }
7317 }
7318 }
7319 }

```

The following macro is for the case of a \Block which uses the key p.

```

7320 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7321 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7322 {
7323     \seq_gput_right:Ne \g_@@_blocks_seq
7324     {
7325         \l_tmpa_tl
7326         { \exp_not:n { #3 } }
7327         {
7328             \group_begin:
7329             \exp_not:n { #4 #5 }
7330             \group_end:
7331         }
7332     }
7333 }

```

The following macro is for the case of a \Block which uses the key p.

```

7334 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7335 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7336 {
7337     \seq_gput_right:Ne \g_@@_blocks_seq
7338     {

```

```

7339     \l_tmpa_tl
7340     { \exp_not:n { #3 } }
7341     { \exp_not:n { #4 #5 } }
7342   }
7343 }

```

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).

```

7344 \keys_define:nn { nicematrix / Block / SecondPass }
7345 {
7346   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7347   ampersand-in-blocks .default:n = true ,
7348   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7349   tikz .code:n =
7350     \IfPackageLoadedTF { tikz }
7351     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7352     { \@@_error:n { tikz~key~without~tikz } } ,
7353   tikz .value_required:n = true ,
7354   fill .code:n =
7355     \tl_set_rescan:Nnn
7356     \l_@@_fill_tl
7357     { \char_set_catcode_other:N ! }
7358     { #1 } ,
7359   fill .value_required:n = true ,
7360   opacity .tl_set:N = \l_@@_opacity_tl ,
7361   opacity .value_required:n = true ,
7362   draw .code:n =
7363     \tl_set_rescan:Nnn
7364     \l_@@_draw_tl
7365     { \char_set_catcode_other:N ! }
7366     { #1 } ,
7367   draw .default:n = default ,
7368   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7369   rounded-corners .default:n = 4 pt ,
7370   color .code:n =
7371     \@@_color:n { #1 }
7372     \tl_set_rescan:Nnn
7373     \l_@@_draw_tl
7374     { \char_set_catcode_other:N ! }
7375     { #1 } ,
7376   borders .clist_set:N = \l_@@_borders_clist ,
7377   borders .value_required:n = true ,
7378   hvlines .meta:n = { vlines , hlines } ,
7379   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7380   vlines .default:n = true ,
7381   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7382   hlines .default:n = true ,
7383   line-width .dim_set:N = \l_@@_line_width_dim ,
7384   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7385   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7386     \bool_set_true:N \l_@@_p_block_bool ,
7387   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7388   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7389   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7390   L .code:n = \str_set:Nn \l_@@_hpos_block_str L
7391     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7392   R .code:n = \str_set:Nn \l_@@_hpos_block_str R
7393     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,

```

```

7394 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7395         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7396 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7397 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7398 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7399 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7400 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7401 m .value_forbidden:n = true ,
7402 v-center .meta:n = m ,
7403 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7404 p .value_forbidden:n = true ,
7405 name .tl_set:N = \l_@@_block_name_str ,
7406 name .value_required:n = true ,
7407 name .initial:n = ,
7408 respect-arraystretch .code:n =
7409     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7410 respect-arraystretch .value_forbidden:n = true ,
7411 transparent .bool_set:N = \l_@@_transparent_bool ,
7412 transparent .default:n = true ,
7413 transparent .initial:n = false ,
7414 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7415 }

```

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.

```

7416 \cs_new_protected:Npn \@@_draw_blocks:
7417 {
7418     \bool_if:NTF \c_@@_tagging_array_bool
7419     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7420     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7421     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7422 }
7423 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7424 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7425 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7426 \int_zero_new:N \l_@@_last_row_int
7427 \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

7428 \int_compare:nNnTF { #3 } > { 99 }
7429 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7430 { \int_set:Nn \l_@@_last_row_int { #3 } }
7431 \int_compare:nNnTF { #4 } > { 99 }
7432 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7433 { \int_set:Nn \l_@@_last_col_int { #4 } }
7434 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7435 {
7436     \bool_lazy_and:nnTF
7437     \l_@@_preamble_bool
7438     {
7439         \int_compare_p:n
7440         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7441     }

```

```

7442     {
7443         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7444         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7445         \@@_msg_redirect_name:nn { columns-not-used } { none }
7446     }
7447     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7448 }
7449 {
7450     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7451     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7452     {
7453         \@@_Block_v:nneenn
7454         { #1 }
7455         { #2 }
7456         { \int_use:N \l_@@_last_row_int }
7457         { \int_use:N \l_@@_last_col_int }
7458         { #5 }
7459         { #6 }
7460     }
7461 }
7462 }

```

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

```

7463 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7464 {

```

The group is for the keys.

```

7465     \group_begin:
7466     \int_compare:nNnT { #1 } = { #3 }
7467     { \str_set:Nn \l_@@_vpos_block_str { t } }
7468     \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`.

```

7469     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7470
7471     \bool_lazy_and:nnT
7472     \l_@@_vlines_block_bool
7473     { ! \l_@@_ampersand_bool }
7474     {
7475         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7476         {
7477             \@@_vlines_block:nnn
7478             { \exp_not:n { #5 } }
7479             { #1 - #2 }
7480             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7481         }
7482     }
7483     \bool_if:NT \l_@@_hlines_block_bool
7484     {
7485         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7486         {
7487             \@@_hlines_block:nnn
7488             { \exp_not:n { #5 } }
7489             { #1 - #2 }
7490             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7491         }
7492     }
7493     \bool_if:NF \l_@@_transparent_bool
7494     {
7495         \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool

```

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).

```

7496         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7497         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7498     }
7499 }

```

```

7500 \tl_if_empty:NF \l_@@_draw_tl
7501 {
7502     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7503     { \@@_error:n { hlines~with~color } }
7504     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7505     {
7506         \@@_stroke_block:nnn

```

#5 are the options

```

7507         { \exp_not:n { #5 } }
7508         { #1 - #2 }
7509         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7510     }
7511     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7512     { { #1 } { #2 } { #3 } { #4 } }
7513 }
7514 \clist_if_empty:NF \l_@@_borders_clist
7515 {
7516     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7517     {
7518         \@@_stroke_borders_block:nnn
7519         { \exp_not:n { #5 } }
7520         { #1 - #2 }
7521         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7522     }
7523 }
7524 \tl_if_empty:NF \l_@@_fill_tl
7525 {
7526     \tl_if_empty:NF \l_@@_opacity_tl
7527     {
7528         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7529             {
7530                 \tl_set:Ne \l_@@_fill_tl
7531                 {
7532                     [ opacity = \l_@@_opacity_tl ,
7533                     \tl_tail:o \l_@@_fill_tl
7534                 }
7535             }
7536             {
7537                 \tl_set:Ne \l_@@_fill_tl
7538                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7539             }
7540         }
7541         \tl_gput_right:Ne \g_@@_pre_code_before_tl
7542         {
7543             \exp_not:N \roundedrectanglecolor
7544             \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7545                 { \l_@@_fill_tl }
7546                 { { \l_@@_fill_tl } }
7547             ]
7548             { #1 - #2 }
7549             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7550             { \dim_use:N \l_@@_rounded_corners_dim }
7551         }

```



```

7552 \seq_if_empty:NF \l_@@_tikz_seq
7553 {
7554   \tl_gput_right:Ne \g_nicematrix_code_before_tl
7555   {
7556     \@@_block_tikz:nnnnn
7557     { \seq_use:Nn \l_@@_tikz_seq { , } }
7558     { #1 }
7559     { #2 }
7560     { \int_use:N \l_@@_last_row_int }
7561     { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of list of Tikz keys.

```

7562   }
7563 }

7564 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7565 {
7566   \tl_gput_right:Ne \g_@@_pre_code_after_tl
7567   {
7568     \@@_actually_diagbox:nnnnnn
7569     { #1 }
7570     { #2 }
7571     { \int_use:N \l_@@_last_row_int }
7572     { \int_use:N \l_@@_last_col_int }
7573     { \exp_not:n { ##1 } }
7574     { \exp_not:n { ##2 } }
7575   }
7576 }

```

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
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7577 \pgfpicture
7578 \pgfrememberpicturepositiononpagetrue
7579 \pgf@relevantforpicturesizefalse
7580 \@@_qpoint:n { row - #1 }
7581 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7582 \@@_qpoint:n { col - #2 }
7583 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7584 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7585 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7586 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7587 \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.

```

7588 \@@pgf_rect_node:nnnnn
7589 { \@@_env: - #1 - #2 - block }
7590 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7591 \str_if_empty:NF \l_@@_block_name_str
7592 {
7593   \pgfnodealias
7594   { \@@_env: - \l_@@_block_name_str }
7595   { \@@_env: - #1 - #2 - block }
7596   \str_if_empty:NF \l_@@_name_str
7597   {
7598     \pgfnodealias
7599     { \l_@@_name_str - \l_@@_block_name_str }
7600     { \@@_env: - #1 - #2 - block }
7601   }
7602 }

```

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.

```

7603 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7604 {
7605   \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.

```

7606 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7607 {

```

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.

```

7608 \cs_if_exist:cT
7609 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7610 {
7611   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7612   {
7613     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7614     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7615   }
7616 }
7617 }

```

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.

```

7618 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7619 {
7620   \@@_qpoint:n { col - #2 }
7621   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7622 }
7623 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7624 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7625 {
7626   \cs_if_exist:cT
7627   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7628   {
7629     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7630     {
7631       \pgfpointanchor
7632       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7633       { east }

```

```

7634         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7635     }
7636 }
7637 }
7638 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7639 {
7640     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7641     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7642 }
7643 \@@_pgf_rect_node:nnnnn
7644 { \@@_env: - #1 - #2 - block - short }
7645 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7646 }

```

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.

```

7647 \bool_if:NT \l_@@_medium_nodes_bool
7648 {
7649     \@@_pgf_rect_node:nnn
7650     { \@@_env: - #1 - #2 - block - medium }
7651     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7652     {
7653         \pgfpointanchor
7654         { \@@_env:
7655             - \int_use:N \l_@@_last_row_int
7656             - \int_use:N \l_@@_last_col_int - medium
7657         }
7658         { south-east }
7659     }
7660 }
7661 \endpgfpicture

7662 \bool_if:NTF \l_@@_ampersand_bool
7663 {
7664     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7665     \int_zero_new:N \l_@@_split_int
7666     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7667     \pgfpicture
7668     \pgfrememberpicturerepositiononpagetrue
7669     \pgf@relevantforpicturesizefalse
7670     \@@_qpoint:n { row - #1 }
7671     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7672     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7673     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7674     \@@_qpoint:n { col - #2 }
7675     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7676     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7677     \dim_set:Nn \l_tmpb_dim
7678     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7679     \bool_lazy_or:nnT
7680     \l_@@_vlines_block_bool
7681     { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7682     {
7683         \int_step_inline:nn { \l_@@_split_int - 1 }
7684         {
7685             \pgfpathmoveto
7686             {
7687                 \pgfpoint
7688                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7689                 \l_@@_tmpc_dim
7690             }
7691             \pgfpathlineto
7692             {

```

```

7693         \pgfpoint
7694         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7695         \l_@@_tmpd_dim
7696     }
7697     \CT@arc@
7698     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7699     \pgfsetrectcap
7700     \pgfusepathqstroke
7701 }
7702 }
7703 \@@_qpoint:n { row - #1 - base }
7704 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7705 \int_step_inline:nn \l_@@_split_int
7706 {
7707     \group_begin:
7708     \dim_set:Nn \col@sep
7709     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7710     \pgftransformshift
7711     {
7712         \pgfpoint
7713         {
7714             \str_case:on \l_@@_hpos_block_str
7715             {
7716                 l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep }
7717                 c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7718                 r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7719             }
7720         }
7721         { \l_@@_tmpc_dim }
7722     }
7723     \pgfset
7724     {
7725         inner~xsep = \c_zero_dim ,
7726         inner~ysep = \c_zero_dim
7727     }
7728     \pgfnode
7729     { rectangle }
7730     {
7731         \str_case:on \l_@@_hpos_block_str
7732         {
7733             c { base }
7734             l { base~west }
7735             r { base~east }
7736         }
7737     }
7738     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7739     \group_end:
7740 }
7741 \endpgfpicture
7742 }

```

Now the case where there is no ampersand & in the content of the block.

```

7743 {
7744     \bool_if:NTF \l_@@_p_block_bool
7745     {

```

When the final user has used the key p, we have to compute the width.

```

7746     \pgfpicture
7747     \pgfrememberpicturepositiononpagetrue
7748     \pgf@relevantforpicturesizefalse
7749     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7750     {
7751         \@@_qpoint:n { col - #2 }
7752         \dim_gset_eq:NN \g_tmpa_dim \pgf@x

```

```

7753         \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7754     }
7755     {
7756         \pgfpointanchor { \l_@@_env: - #1 - #2 - block - short } { west }
7757         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7758         \pgfpointanchor { \l_@@_env: - #1 - #2 - block - short } { east }
7759     }
7760     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7761     \endpgfpicture
7762     \hbox_set:Nn \l_@@_cell_box
7763     {
7764         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7765         { \g_tmpb_dim }
7766         \str_case:on \l_@@_hpos_block_str
7767         { c \centering r \raggedleft l \raggedright j { } }
7768         #6
7769         \end { minipage }
7770     }
7771 }
7772 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7773 \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.

```

7774     \pgfpicture
7775     \pgfrememberpicturepositiononpagetrue
7776     \pgf@relevantforpicturesizefalse
7777     \bool_lazy_any:nTF
7778     {
7779         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7780         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7781         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7782         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7783     }
7784     {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7785         \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`.

```

7786         \bool_if:nT \g_@@_last_col_found_bool
7787         {
7788             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7789             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7790         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7791         \tl_set:Ne \l_tmpa_tl
7792         {
7793             \str_case:on \l_@@_vpos_block_str
7794             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7795             { } { % added 2024-06-29
7796                 \str_case:on \l_@@_hpos_block_str
7797                 {
7798                     c { center }
7799                     l { west }
7800                     r { east }
7801                     j { center }
7802                 }
7803             }

```

```

7804         c {
7805             \str_case:on \l_@@_hpos_block_str
7806             {
7807                 c { center }
7808                 l { west }
7809                 r { east }
7810                 j { center }
7811             }
7812         }
7813     T {
7814         \str_case:on \l_@@_hpos_block_str
7815         {
7816             c { north }
7817             l { north-west }
7818             r { north-east }
7819             j { north }
7820         }
7821     }
7822 }
7823 B {
7824     \str_case:on \l_@@_hpos_block_str
7825     {
7826         c { south }
7827         l { south-west }
7828         r { south-east }
7829         j { south }
7830     }
7831 }
7832 }
7833 }
7834 }
7835 }
7836 \pgftransformshift
7837 {
7838     \pgfpointanchor
7839     {
7840         \@@_env: - #1 - #2 - block
7841         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7842     }
7843     { \l_tmpa_tl }
7844 }
7845 \pgfset
7846 {
7847     inner-xsep = \c_zero_dim ,
7848     inner-ysep = \c_zero_dim
7849 }
7850 \pgfnode
7851 { rectangle }
7852 { \l_tmpa_tl }
7853 { \box_use_drop:N \l_@@_cell_box } { } { }
7854 }

```

End of the case when $\l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7855 {
7856     \pgfextracty \l_tmpa_dim
7857     {
7858         \@@_qpoint:n
7859         {
7860             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7861             - base
7862         }
7863     }
7864     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7865     \pgfpointanchor
7866     {
7867       \@@_env: - #1 - #2 - block
7868       \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7869     }
7870     {
7871       \str_case:on \l_@@_hpos_block_str
7872       {
7873         c { center }
7874         l { west }
7875         r { east }
7876         j { center }
7877       }
7878     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7879     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7880     \pgfset { inner~sep = \c_zero_dim }
7881     \pgfnode
7882     { rectangle }
7883     {
7884       \str_case:on \l_@@_hpos_block_str
7885       {
7886         c { base }
7887         l { base~west }
7888         r { base~east }
7889         j { base }
7890       }
7891     }
7892     { \box_use_drop:N \l_@@_cell_box } { } { }
7893   }
7894   \endpgfpicture
7895 }
7896 \group_end:
7897 }

```

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).

```

7898 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7899 {
7900   \group_begin:
7901   \tl_clear:N \l_@@_draw_tl
7902   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7903   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
7904   \pgfpicture
7905   \pgfrememberpicturepositiononpagetrue
7906   \pgf@relevantforpicturesizefalse
7907   \tl_if_empty:NF \l_@@_draw_tl
7908   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7909     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7910     { \CT@arc@ }
7911     { \@@_color:o \l_@@_draw_tl }
7912   }
7913   \pgfsetcornersarced
7914   {
7915     \pgfpoint
7916     { \l_@@_rounded_corners_dim }
7917     { \l_@@_rounded_corners_dim }

```

```

7918     }
7919     \@@_cut_on_hyphen:w #2 \q_stop
7920     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7921     {
7922         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7923         {
7924             \@@_qpoint:n { row - \l_tmpa_tl }
7925             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7926             \@@_qpoint:n { col - \l_tmpb_tl }
7927             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7928             \@@_cut_on_hyphen:w #3 \q_stop
7929             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7930             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7931             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7932             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7933             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7934             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7935             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7936             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7937             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7938             \pgfpathrectanglecorners
7939             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7940             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7941             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7942             { \pgfusepathqstroke }
7943             { \pgfusepath { stroke } }
7944         }
7945     }
7946     \endpgfpicture
7947     \group_end:
7948 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7949 \keys_define:nn { nicematrix / BlockStroke }
7950 {
7951     color .tl_set:N = \l_@@_draw_tl ,
7952     draw .code:n =
7953     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7954     draw .default:n = default ,
7955     line-width .dim_set:N = \l_@@_line_width_dim ,
7956     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7957     rounded-corners .default:n = 4 pt
7958 }

```

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).

```

7959 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7960 {
7961     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7962     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7963     \@@_cut_on_hyphen:w #2 \q_stop
7964     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7965     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7966     \@@_cut_on_hyphen:w #3 \q_stop
7967     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7968     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7969     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7970     {
7971         \use:e
7972         {
7973             \@@_vline:n
7974             {

```



```

7975         position = ##1 ,
7976         start = \l_@@_tmpc_tl ,
7977         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7978         total-width = \dim_use:N \l_@@_line_width_dim
7979     }
7980 }
7981 }
7982 }
7983 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7984 {
7985     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7986     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7987     \@@_cut_on_hyphen:w #2 \q_stop
7988     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7989     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7990     \@@_cut_on_hyphen:w #3 \q_stop
7991     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7992     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7993     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7994     {
7995         \use:e
7996         {
7997             \@@_hline:n
7998             {
7999                 position = ##1 ,
8000                 start = \l_@@_tmpd_tl ,
8001                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
8002                 total-width = \dim_use:N \l_@@_line_width_dim
8003             }
8004         }
8005     }
8006 }

```

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).

```

8007 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8008 {
8009     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8010     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8011     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8012     { \@@_error:n { borders~forbidden } }
8013     {
8014         \tl_clear_new:N \l_@@_borders_tikz_tl
8015         \keys_set:no
8016         { nicematrix / OnlyForTikzInBorders }
8017         \l_@@_borders_clist
8018         \@@_cut_on_hyphen:w #2 \q_stop
8019         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8020         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8021         \@@_cut_on_hyphen:w #3 \q_stop
8022         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8023         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8024         \@@_stroke_borders_block_i:
8025     }
8026 }
8027 \hook_gput_code:nnn { begindocument } { . }
8028 {
8029     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8030     {
8031         \c_@@_pgfortikzpicture_tl
8032         \@@_stroke_borders_block_ii:

```

```

8033     \c_@@_endpgfortikzpicture_tl
8034   }
8035 }

8036 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8037 {
8038   \pgfrememberpicturepositiononpagetrue
8039   \pgf@relevantforpicturesizefalse
8040   \CT@arc@
8041   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8042   \clist_if_in:NnT \l_@@_borders_clist { right }
8043   { \@@_stroke_vertical:n \l_tmpb_tl }
8044   \clist_if_in:NnT \l_@@_borders_clist { left }
8045   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8046   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8047   { \@@_stroke_horizontal:n \l_tmpa_tl }
8048   \clist_if_in:NnT \l_@@_borders_clist { top }
8049   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8050 }

8051 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8052 {
8053   tikz .code:n =
8054     \cs_if_exist:NTF \tikzpicture
8055     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8056     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8057   tikz .value_required:n = true ,
8058   top .code:n = ,
8059   bottom .code:n = ,
8060   left .code:n = ,
8061   right .code:n = ,
8062   unknown .code:n = \@@_error:n { bad~border }
8063 }

```

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).

```

8064 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8065 {
8066   \@@_qpoint:n \l_@@_tmpc_tl
8067   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8068   \@@_qpoint:n \l_tmpa_tl
8069   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8070   \@@_qpoint:n { #1 }
8071   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8072   {
8073     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8074     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8075     \pgfusepath{stroke}
8076   }
8077   {
8078     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8079     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8080   }
8081 }

```

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).

```

8082 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8083 {
8084   \@@_qpoint:n \l_@@_tmpd_tl
8085   \clist_if_in:NnTF \l_@@_borders_clist { left }
8086   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8087   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8088   \@@_qpoint:n \l_tmpb_tl

```

```

8089 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8090 \@@_qpoint:n { #1 }
8091 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8092 {
8093   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8094   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8095   \pgfusepathqstroke
8096 }
8097 {
8098   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8099   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8100 }
8101 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8102 \keys_define:nn { nicematrix / BlockBorders }
8103 {
8104   borders .clist_set:N = \l_@@_borders_clist ,
8105   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8106   rounded-corners .default:n = 4 pt ,
8107   line-width .dim_set:N = \l_@@_line_width_dim
8108 }

```

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.

```

8109 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8110 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8111 {
8112   \begin { tikzpicture }
8113   \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because **#5** is a list of lists.

```

8114   \clist_map_inline:nn { #1 }
8115   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8116   \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8117   \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8118   (
8119     [
8120       xshift = \dim_use:N \l_@@_offset_dim ,
8121       yshift = - \dim_use:N \l_@@_offset_dim
8122     ]
8123     #2 -| #3
8124   )
8125   rectangle
8126   (
8127     [
8128       xshift = - \dim_use:N \l_@@_offset_dim ,
8129       yshift = \dim_use:N \l_@@_offset_dim
8130     ]
8131     \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8132   ) ;
8133 }
8134 \end { tikzpicture }
8135 }

```

```

8136 \keys_define:nn { nicematrix / SpecialOffset }
8137 { 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.

```

8138 \cs_new_protected:Npn \@@_NullBlock:
8139 { \@@_collect_options:n { \@@_NullBlock_i: } }
8140 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8141 { }

```

27 How to draw the dotted lines transparently

```

8142 \cs_set_protected:Npn \@@_renew_matrix:
8143 {
8144   \RenewDocumentEnvironment { pmatrix } { } {
8145     { \pNiceMatrix }
8146     { \endpNiceMatrix }
8147   \RenewDocumentEnvironment { vmatrix } { } {
8148     { \vNiceMatrix }
8149     { \endvNiceMatrix }
8150   \RenewDocumentEnvironment { Vmatrix } { } {
8151     { \VNiceMatrix }
8152     { \endVNiceMatrix }
8153   \RenewDocumentEnvironment { bmatrix } { } {
8154     { \bNiceMatrix }
8155     { \endbNiceMatrix }
8156   \RenewDocumentEnvironment { Bmatrix } { } {
8157     { \BNiceMatrix }
8158     { \endBNiceMatrix }
8159 }

```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8160 \keys_define:nn { nicematrix / Auto }
8161 {
8162   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8163   columns-type .value_required:n = true ,
8164   l .meta:n = { columns-type = l } ,
8165   r .meta:n = { columns-type = r } ,
8166   c .meta:n = { columns-type = c } ,
8167   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8168   delimiters / color .value_required:n = true ,
8169   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8170   delimiters / max-width .default:n = true ,
8171   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8172   delimiters .value_required:n = true ,
8173   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8174   rounded-corners .default:n = 4 pt
8175 }
8176 \NewDocumentCommand \AutoNiceMatrixWithDelims
8177 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8178 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8179 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8180 {

```

The group is for the protection of the keys.

```

8181 \group_begin:
8182 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8183 \use:e
8184 {
8185   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8186   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8187   [ \exp_not:o \l_tmpa_tl ]
8188 }
8189 \int_if_zero:nT \l_@@_first_row_int
8190 {
8191   \int_if_zero:nT \l_@@_first_col_int { & }
8192   \prg_replicate:nn { #4 - 1 } { & }
8193   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8194 }
8195 \prg_replicate:nn { #3 }
8196 {
8197   \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).

```

8198   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8199   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8200 }
8201 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8202 {
8203   \int_if_zero:nT \l_@@_first_col_int { & }
8204   \prg_replicate:nn { #4 - 1 } { & }
8205   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8206 }
8207 \end { NiceArrayWithDelims }
8208 \group_end:
8209 }

8210 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8211 {
8212   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8213   {
8214     \bool_gset_true:N \g_@@_delims_bool
8215     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8216     \AutoNiceMatrixWithDelims { #2 } { #3 }
8217   }
8218 }

8219 \@@_define_com:nnn p ( )
8220 \@@_define_com:nnn b [ ]
8221 \@@_define_com:nnn v | |
8222 \@@_define_com:nnn V \l \l
8223 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8224 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8225 {
8226   \group_begin:
8227   \bool_gset_false:N \g_@@_delims_bool
8228   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8229   \group_end:
8230 }

```

29 The redefinition of the command `\dotfill`

```

8231 \cs_set_eq:NN \@@_old_dotfill \dotfill
8232 \cs_new_protected:Npn \@@_dotfill:
8233 {

```

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}`).

```

8234 \@@_old_dotfill
8235 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8236 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8237 \cs_new_protected:Npn \@@_dotfill_i:
8238 { \dim_compare:nNtT { \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.

```

8239 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8240 {
8241 \tl_gput_right:Ne \g_@@_pre_code_after_tl
8242 {
8243 \@@_actually_diagbox:nnnnnn
8244 { \int_use:N \c@iRow }
8245 { \int_use:N \c@jCol }
8246 { \int_use:N \c@iRow }
8247 { \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.

```

8248 { \g_@@_row_style_tl \exp_not:n { #1 } }
8249 { \g_@@_row_style_tl \exp_not:n { #2 } }
8250 }

```

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`.

```

8251 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8252 {
8253 { \int_use:N \c@iRow }
8254 { \int_use:N \c@jCol }
8255 { \int_use:N \c@iRow }
8256 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8257 { }
8258 }
8259 }

```

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.

```

8260 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8261 {

```

```

8262 \pgfpicture
8263 \pgf@relevantforpicturesizefalse
8264 \pgfrememberpicturepositiononpagetrue
8265 \@@_qpoint:n { row - #1 }
8266 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8267 \@@_qpoint:n { col - #2 }
8268 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8269 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8270 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8271 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8272 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8273 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8274 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8275 {

```

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.

```

8276 \CT@arc@
8277 \pgfsetroundcap
8278 \pgfusepathqstroke
8279 }
8280 \pgfset { inner~sep = 1 pt }
8281 \pgfscope
8282 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8283 \pgfnode { rectangle } { south~west }
8284 {
8285 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8286 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8287 \end { minipage }
8288 }
8289 { }
8290 { }
8291 \endpgfscope
8292 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8293 \pgfnode { rectangle } { north~east }
8294 {
8295 \begin { minipage } { 20 cm }
8296 \raggedleft
8297 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8298 \end { minipage }
8299 }
8300 { }
8301 { }
8302 \endpgfpicture
8303 }

```

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`.

```

8304 \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 `\`.

```
8305 \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`.

```
8306 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8307 {
8308   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8309   \@@_CodeAfter_iv:n
8310 }
```

We catch the argument of the command `\end` (in `#1`).

```
8311 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8312 {
```

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.

```
8313   \str_if_eq:eeTF \currenvir { #1 }
8314   { \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`.

```
8315   {
8316     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8317     \@@_CodeAfter_ii:n
8318   }
8319 }
```

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.

```
8320 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8321 {
8322   \pgfpicture
8323   \pgfrememberpicturepositiononpagetrue
8324   \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.

```
8325   \@@_qpoint:n { row - 1 }
8326   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8327   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8328   \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).

```
8329   \bool_if:nTF { #3 }
8330   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8331   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
```



```

8332 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8333 {
8334   \cs_if_exist:cT
8335   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8336   {
8337     \pgfpointanchor
8338     { \@@_env: - ##1 - #2 }
8339     { \bool_if:nTF { #3 } { west } { east } }
8340     \dim_set:Nn \l_tmpa_dim
8341     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8342   }
8343 }

```

Now we can put the delimiter with a node of PGF.

```

8344 \pgfset { inner~sep = \c_zero_dim }
8345 \dim_zero:N \nulldelimiterspace
8346 \pgftransformshift
8347 {
8348   \pgfpoint
8349   { \l_tmpa_dim }
8350   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8351 }
8352 \pgfnode
8353 { rectangle }
8354 { \bool_if:nTF { #3 } { east } { west } }
8355 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8356 \nullfont
8357 \c_math_toggle_token
8358 \@@_color:o \l_@@_delimiters_color_tl
8359 \bool_if:nTF { #3 } { \left #1 } { \left . }
8360 \vcenter
8361 {
8362   \nullfont
8363   \hrule \@height
8364   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8365   \@depth \c_zero_dim
8366   \@width \c_zero_dim
8367 }
8368 \bool_if:nTF { #3 } { \right . } { \right #1 }
8369 \c_math_toggle_token
8370 }
8371 { }
8372 { }
8373 \endpgfpicture
8374 }

```

33 The command `\SubMatrix`

```

8375 \keys_define:nn { nicematrix / sub-matrix }
8376 {
8377   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8378   extra-height .value_required:n = true ,
8379   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8380   left-xshift .value_required:n = true ,
8381   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8382   right-xshift .value_required:n = true ,
8383   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8384   xshift .value_required:n = true ,
8385   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

8386     delimiters / color .value_required:n = true ,
8387     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8388     slim .default:n = true ,
8389     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8390     hlines .default:n = all ,
8391     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8392     vlines .default:n = all ,
8393     hvlines .meta:n = { hlines, vlines } ,
8394     hvlines .value_forbidden:n = true
8395   }
8396 \keys_define:nn { nicematrix }
8397 {
8398   SubMatrix .inherit:n = nicematrix / sub-matrix ,
8399   NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8400   pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8401   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8402 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8403 \keys_define:nn { nicematrix / SubMatrix }
8404 {
8405   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8406   delimiters / color .value_required:n = true ,
8407   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8408   hlines .default:n = all ,
8409   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8410   vlines .default:n = all ,
8411   hvlines .meta:n = { hlines, vlines } ,
8412   hvlines .value_forbidden:n = true ,
8413   name .code:n =
8414     \tl_if_empty:nTF { #1 }
8415     { \@@_error:n { Invalid-name } }
8416     {
8417       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8418       {
8419         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8420         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8421         {
8422           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8423           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8424         }
8425       }
8426       { \@@_error:n { Invalid-name } }
8427     } ,
8428   name .value_required:n = true ,
8429   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8430   rules .value_required:n = true ,
8431   code .tl_set:N = \l_@@_code_tl ,
8432   code .value_required:n = true ,
8433   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8434 }

8435 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8436 {
8437   \peek_remove_spaces:n
8438   {
8439     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8440     {
8441       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8442       [
8443         delimiters / color = \l_@@_delimiters_color_tl ,
8444         hlines = \l_@@_submatrix_hlines_clist ,

```

```

8445         vlines = \l_@@_submatrix_vlines_clist ,
8446         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8447         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8448         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8449         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8450         #5
8451     ]
8452 }
8453 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8454 }
8455 }
8456 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8457 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8458 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8459 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8460 {
8461     \seq_gput_right:Ne \g_@@_submatrix_seq
8462     {
We use \str_if_eq:eeTF because it is fully expandable (and slightly faster than \tl_if_eq:nnTF).
8463         { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8464         { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8465         { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8466         { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8467     }
8468 }

```

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`.

```

8469 \hook_gput_code:nnn { begindocument } { . }
8470 {
8471     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8472     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8473     \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8474     {
8475         \peek_remove_spaces:n
8476         {
8477             \@@_sub_matrix:nnnnnnn
8478             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8479         }
8480     }
8481 }

```

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).

```

8482 \NewDocumentCommand \@@_compute_i_j:nn
8483 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8484 { \@@_compute_i_j:nnnn #1 #2 }

```

```

8485 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8486 {
8487   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8488   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8489   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8490   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8491   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8492     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8493   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8494     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8495   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8496     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8497   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8498     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8499 }
8500 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8501 {
8502   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8503   \@@_compute_i_j:nn { #2 } { #3 }
8504   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8505     { \cs_set_nopar:Npn \arraystretch { 1 } }
8506   \bool_lazy_or:nnTF
8507     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8508     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8509     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8510   {
8511     \str_clear_new:N \l_@@_submatrix_name_str
8512     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8513     \pgfpicture
8514     \pgfrememberpicturepositiononpagetrue
8515     \pgf@relevantforpicturesizefalse
8516     \pgfset { inner~sep = \c_zero_dim }
8517     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8518     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

8519   \bool_if:NTF \l_@@_submatrix_slim_bool
8520     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8521     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8522   {
8523     \cs_if_exist:cT
8524       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8525     {
8526       \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8527       \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8528         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8529     }
8530     \cs_if_exist:cT
8531       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8532     {
8533       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8534       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8535         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8536     }
8537   }
8538   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8539     { \@@_error:nn { Impossible~delimiter } { left } }
8540   {
8541     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8542       { \@@_error:nn { Impossible~delimiter } { right } }
8543       { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8544   }

```

```

8545     \endpgfpicture
8546   }
8547 \group_end:
8548 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8549 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8550 {
8551   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8552   \dim_set:Nn \l_@@_y_initial_dim
8553   {
8554     \fp_to_dim:n
8555     {
8556       \pgf@y
8557       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8558     }
8559   }
8560   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8561   \dim_set:Nn \l_@@_y_final_dim
8562   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8563   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8564   {
8565     \cs_if_exist:cT
8566     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8567     {
8568       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8569       \dim_set:Nn \l_@@_y_initial_dim
8570       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8571     }
8572     \cs_if_exist:cT
8573     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8574     {
8575       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8576       \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8577       { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8578     }
8579   }
8580   \dim_set:Nn \l_tmpa_dim
8581   {
8582     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8583     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8584   }
8585   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8586   \group_begin:
8587   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8588   \@@_set_CT@arc@:o \l_@@_rules_color_tl
8589   \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`.

```

8590   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8591   {
8592     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8593     {
8594       \int_compare:nNnT
8595       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8596       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8597     \@@_qpoint:n { col - ##1 }

```

```

8598         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8599         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8600         \pgfusepathqstroke
8601     }
8602 }
8603 }

```

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.

```

8604 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8605 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8606 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8607 {
8608     \bool_lazy_and:nnTF
8609     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8610     {
8611         \int_compare_p:nNn
8612         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8613     {
8614         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8615         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8616         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8617         \pgfusepathqstroke
8618     }
8619     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8620 }

```

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.

```

8621 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8622 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8623 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8624 {
8625     \bool_lazy_and:nnTF
8626     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8627     {
8628         \int_compare_p:nNn
8629         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8630     {
8631         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8632     \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8633     \dim_set:Nn \l_tmpa_dim
8634     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8635     \str_case:nn { #1 }
8636     {
8637         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8638         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8639         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8640     }
8641     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8642     \dim_set:Nn \l_tmpb_dim
8643     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8644     \str_case:nn { #2 }
8645     {
8646         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8647         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8648         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8649     }

```

```

8650         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8651         \pgfusepathqstroke
8652     \group_end:
8653 }
8654 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
8655 }

```

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).

```

8656 \str_if_empty:NF \l_@@_submatrix_name_str
8657 {
8658     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8659     \l_@@_x_initial_dim \l_@@_y_initial_dim
8660     \l_@@_x_final_dim \l_@@_y_final_dim
8661 }
8662 \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`.

```

8663 \begin { pgfscope }
8664 \pgftransformshift
8665 {
8666     \pgfpoint
8667     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8668     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8669 }
8670 \str_if_empty:NTF \l_@@_submatrix_name_str
8671 { \@@_node_left:nn #1 { } }
8672 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8673 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8674 \pgftransformshift
8675 {
8676     \pgfpoint
8677     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8678     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8679 }
8680 \str_if_empty:NTF \l_@@_submatrix_name_str
8681 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8682 {
8683     \@@_node_right:nnnn #2
8684     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8685 }
8686 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8687 \flag_clear_new:N \l_@@_code_flag
8688 \l_@@_code_tl
8689 }

```

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 $i-lj$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8690 \cs_set_eq:NN \@@_old_pgfpntanchor \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.

```

8691 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8692 {
8693   \use:e
8694   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8695 }

```

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 -.

```

8696 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8697 { #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.

```

8698 \tl_const:Nn \c_@@_integers_alist_tl
8699 {
8700   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8701   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8702   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8703   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8704 }

8705 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8706 {

```

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`.

```

8707   \tl_if_empty:nTF { #2 }
8708   {
8709     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8710     {
8711       \flag_raise:N \l_@@_code_flag
8712       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8713       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8714       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8715     }
8716     { #1 }
8717   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8718     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8719   }

```

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`).

```

8720 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8721 {
8722   \str_case:nnF { #1 }
8723   {
8724     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8725     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8726   }

```

Now the case of a node of the form $i-j$.

```

8727   {
8728     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8729     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8730   }
8731 }

```


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`).

```

8732 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8733 {
8734   \pgfnode
8735     { rectangle }
8736     { east }
8737     {
8738       \nullfont
8739       \c_math_toggle_token
8740       \@@_color:o \l_@@_delimiters_color_tl
8741       \left #1
8742       \vcenter
8743         {
8744           \nullfont
8745           \hrule \@height \l_tmpa_dim
8746             \depth \c_zero_dim
8747             \width \c_zero_dim
8748         }
8749       \right .
8750       \c_math_toggle_token
8751     }
8752     { #2 }
8753     { }
8754 }

```

The command `\@@_node_right:nnn` 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.

```

8755 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8756 {
8757   \pgfnode
8758     { rectangle }
8759     { west }
8760     {
8761       \nullfont
8762       \c_math_toggle_token
8763       \colorlet { current-color } { . }
8764       \@@_color:o \l_@@_delimiters_color_tl
8765       \left .
8766       \vcenter
8767         {
8768           \nullfont
8769           \hrule \@height \l_tmpa_dim
8770             \depth \c_zero_dim
8771             \width \c_zero_dim
8772         }
8773       \right #1
8774       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8775       ^ { \color { current-color } \smash { #4 } }
8776       \c_math_toggle_token
8777     }
8778     { #2 }
8779     { }
8780 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8781 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8782 {
8783   \peek_remove_spaces:n
8784   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8785 }

8786 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8787 {
8788   \peek_remove_spaces:n
8789   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8790 }

8791 \keys_define:nn { nicematrix / Brace }
8792 {
8793   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8794   left-shorten .default:n = true ,
8795   left-shorten .value_forbidden:n = true ,
8796   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8797   right-shorten .default:n = true ,
8798   right-shorten .value_forbidden:n = true ,
8799   shorten .meta:n = { left-shorten , right-shorten } ,
8800   shorten .value_forbidden:n = true ,
8801   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8802   yshift .value_required:n = true ,
8803   yshift .initial:n = \c_zero_dim ,
8804   color .tl_set:N = \l_tmpa_tl ,
8805   color .value_required:n = true ,
8806   unknown .code:n = \@@_error:n { Unknown-key~for~Brace }
8807 }

```

#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`.

```

8808 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8809 {
8810   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8811 \@@_compute_i_j:nn { #1 } { #2 }
8812 \bool_lazy_or:nnTF
8813 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8814 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8815 {
8816   \str_if_eq:eeTF { #5 } { under }
8817   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8818   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8819 }
8820 {
8821   \tl_clear:N \l_tmpa_tl
8822   \keys_set:nn { nicematrix / Brace } { #4 }
8823   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8824   \pgfpicture
8825   \pgfrememberpicturepositiononpagetrue
8826   \pgf@relevantforpicturesizefalse
8827   \bool_if:NT \l_@@_brace_left_shorten_bool
8828   {
8829     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8830     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8831     {

```

```

8832         \cs_if_exist:cT
8833         { \pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_first_j_tl }
8834         {
8835             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_first_j_tl } { west }
8836
8837             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8838             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8839         }
8840     }
8841 }
8842 \bool_lazy_or:nnT
8843 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8844 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8845 {
8846     \l_@@_qpoint:n { col - \l_@@_first_j_tl }
8847     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8848 }
8849 \bool_if:NT \l_@@_brace_right_shorten_bool
8850 {
8851     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8852     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8853     {
8854         \cs_if_exist:cT
8855         { \pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
8856         {
8857             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
8858             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8859             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8860         }
8861     }
8862 }
8863 \bool_lazy_or:nnT
8864 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8865 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8866 {
8867     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8868     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8869 }
8870 \pgfset { inner~sep = \c_zero_dim }
8871 \str_if_eq:eeTF { #5 } { under }
8872 { \l_@@_underbrace_i:n { #3 } }
8873 { \l_@@_overbrace_i:n { #3 } }
8874 \endpgfpicture
8875 }
8876 \group_end:
8877 }

```

The argument is the text to put above the brace.

```

8878 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
8879 {
8880     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
8881     \pgftransformshift
8882     {
8883         \pgfpoint
8884         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8885         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8886     }
8887     \pgfnode
8888     { rectangle }
8889     { south }
8890     {
8891         \vtop
8892         {
8893             \group_begin:

```

```

8894         \everycr { }
8895     \halign
8896     {
8897         \hfil ## \hfil \crcr
8898         \@@_math_toggle: #1 \@@_math_toggle: \cr
8899         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8900         \c_math_toggle_token
8901         \overbrace
8902         {
8903             \hbox_to_wd:nn
8904             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8905             { }
8906         }
8907         \c_math_toggle_token
8908     \cr
8909     }
8910 \group_end:
8911 }
8912 }
8913 { }
8914 { }
8915 }

```

The argument is the text to put under the brace.

```

8916 \cs_new_protected:Npn \@@_underbrace_i:n #1
8917 {
8918     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8919     \pgftransformshift
8920     {
8921         \pgfpoint
8922         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8923         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8924     }
8925     \pgfnode
8926     { rectangle }
8927     { north }
8928     {
8929         \group_begin:
8930         \everycr { }
8931         \vbox
8932         {
8933             \halign
8934             {
8935                 \hfil ## \hfil \crcr
8936                 \c_math_toggle_token
8937                 \underbrace
8938                 {
8939                     \hbox_to_wd:nn
8940                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8941                     { }
8942                 }
8943                 \c_math_toggle_token
8944             \cr
8945             \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8946             \@@_math_toggle: #1 \@@_math_toggle: \cr
8947         }
8948     }
8949     \group_end:
8950 }
8951 { }
8952 { }
8953 }

```

35 The command TikzEveryCell

```

8954 \bool_new:N \l_@@_not_empty_bool
8955 \bool_new:N \l_@@_empty_bool
8956
8957 \keys_define:nn { nicematrix / TikzEveryCell }
8958 {
8959   not-empty .code:n =
8960     \bool_lazy_or:nnTF
8961       \l_@@_in_code_after_bool
8962       \g_@@_recreate_cell_nodes_bool
8963       { \bool_set_true:N \l_@@_not_empty_bool }
8964       { \@@_error:n { detection-of-empty-cells } } ,
8965   not-empty .value_forbidden:n = true ,
8966   empty .code:n =
8967     \bool_lazy_or:nnTF
8968       \l_@@_in_code_after_bool
8969       \g_@@_recreate_cell_nodes_bool
8970       { \bool_set_true:N \l_@@_empty_bool }
8971       { \@@_error:n { detection-of-empty-cells } } ,
8972   empty .value_forbidden:n = true ,
8973   unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
8974 }
8975
8976
8977 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8978 {
8979   \IfPackageLoadedTF { tikz }
8980   {
8981     \group_begin:
8982     \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.

```

8983     \tl_set:Nn \l_tmpa_tl { { #2 } }
8984     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8985     { \@@_for_a_block:nnnnn #1 }
8986     \@@_all_the_cells:
8987     \group_end:
8988   }
8989   { \@@_error:n { TikzEveryCell-without-tikz } }
8990 }
8991
8992 \tl_new:N \@@_i_tl
8993 \tl_new:N \@@_j_tl
8994
8995
8996 \cs_new_protected:Nn \@@_all_the_cells:
8997 {
8998   \int_step_variable:nNn \c@iRow \@@_i_tl
8999   {
9000     \int_step_variable:nNn \c@jCol \@@_j_tl
9001     {
9002       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9003       {
9004         \clist_if_in:NcF \l_@@_corners_cells_clist
9005         { \@@_i_tl - \@@_j_tl }
9006         {
9007           \bool_set_false:N \l_tmpa_bool
9008           \cs_if_exist:cTF
9009           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9010           {
9011             \bool_if:NF \l_@@_empty_bool

```

```

9012         { \bool_set_true:N \l_tmpa_bool }
9013     }
9014     {
9015         \bool_if:NF \l_@@_not_empty_bool
9016         { \bool_set_true:N \l_tmpa_bool }
9017     }
9018     \bool_if:NT \l_tmpa_bool
9019     {
9020         \@@_block_tikz:nnnnn
9021         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9022     }
9023     }
9024     }
9025     }
9026     }
9027 }
9028
9029 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9030 {
9031     \bool_if:NF \l_@@_empty_bool
9032     {
9033         \@@_block_tikz:nnnnn
9034         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9035     }
9036     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9037 }
9038
9039 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9040 {
9041     \int_step_inline:nnn { #1 } { #3 }
9042     {
9043         \int_step_inline:nnn { #2 } { #4 }
9044         { \cs_set_nopar:cpn { cell - ##1 - #####1 } { } }
9045     }
9046 }

```

36 The command \ShowCellNames

```

9047 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9048 {
9049     \dim_gzero_new:N \g_@@_tmpc_dim
9050     \dim_gzero_new:N \g_@@_tmpd_dim
9051     \dim_gzero_new:N \g_@@_tmpe_dim
9052     \int_step_inline:nn \c@iRow
9053     {
9054         \begin { pgfpicture }
9055         \@@_qpoint:n { row - ##1 }
9056         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9057         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9058         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9059         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9060         \bool_if:NTF \l_@@_in_code_after_bool
9061         \end { pgfpicture }
9062         \int_step_inline:nn \c@jCol
9063         {
9064             \hbox_set:Nn \l_tmpa_box
9065             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
9066             \begin { pgfpicture }
9067             \@@_qpoint:n { col - #####1 }
9068             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9069             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9070             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }

```

```

9071 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9072 \endpgfpicture
9073 \end { pgfpicture }
9074 \fp_set:Nn \l_tmpa_fp
9075 {
9076   \fp_min:nn
9077   {
9078     \fp_min:nn
9079     {
9080       \dim_ratio:nn
9081       { \g_@@_tmpd_dim }
9082       { \box_wd:N \l_tmpa_box }
9083     }
9084     {
9085       \dim_ratio:nn
9086       { \g_tmpb_dim }
9087       { \box_ht_plus_dp:N \l_tmpa_box }
9088     }
9089   }
9090   { 1.0 }
9091 }
9092 \box_scale:Nnn \l_tmpa_box
9093 { \fp_use:N \l_tmpa_fp }
9094 { \fp_use:N \l_tmpa_fp }
9095 \pgfpicture
9096 \pgfrememberpicturepositiononpagetrue
9097 \pgf@relevantforpicturesizefalse
9098 \pgftransformshift
9099 {
9100   \pgfpoint
9101   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9102   { \dim_use:N \g_tmpa_dim }
9103 }
9104 \pgfnode
9105 { rectangle }
9106 { center }
9107 { \box_use:N \l_tmpa_box }
9108 { }
9109 { }
9110 \endpgfpicture
9111 }
9112 }
9113 }
9114 \NewDocumentCommand \@@_ShowCellNames { }
9115 {
9116   \bool_if:NT \l_@@_in_code_after_bool
9117   {
9118     \pgfpicture
9119     \pgfrememberpicturepositiononpagetrue
9120     \pgf@relevantforpicturesizefalse
9121     \pgfpathrectanglecorners
9122     { \@@_qpoint:n { 1 } }
9123     {
9124       \@@_qpoint:n
9125       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9126     }
9127     \pgfsetfillopacity { 0.75 }
9128     \pgfsetfillcolor { white }
9129     \pgfusepathqfill
9130     \endpgfpicture
9131   }
9132   \dim_gzero_new:N \g_@@_tmpc_dim
9133   \dim_gzero_new:N \g_@@_tmpd_dim

```

```

9134 \dim_gzero_new:N \g_@@_tmpe_dim
9135 \int_step_inline:nn \c@iRow
9136 {
9137   \bool_if:NTF \l_@@_in_code_after_bool
9138   {
9139     \pgfpicture
9140     \pgfrememberpicturepositiononpagetrue
9141     \pgf@relevantforpicturesizefalse
9142   }
9143   { \begin { pgfpicture } }
9144   \@@_qpoint:n { row - ##1 }
9145   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9146   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9147   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9148   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9149   \bool_if:NTF \l_@@_in_code_after_bool
9150   { \endpgfpicture }
9151   { \end { pgfpicture } }
9152   \int_step_inline:nn \c@jCol
9153   {
9154     \hbox_set:Nn \l_tmpa_box
9155     {
9156       \normalfont \Large \sffamily \bfseries
9157       \bool_if:NTF \l_@@_in_code_after_bool
9158       { \color { red } }
9159       { \color { red ! 50 } }
9160       ##1 - ####1
9161     }
9162     \bool_if:NTF \l_@@_in_code_after_bool
9163     {
9164       \pgfpicture
9165       \pgfrememberpicturepositiononpagetrue
9166       \pgf@relevantforpicturesizefalse
9167     }
9168     { \begin { pgfpicture } }
9169     \@@_qpoint:n { col - ####1 }
9170     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9171     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9172     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9173     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9174     \bool_if:NTF \l_@@_in_code_after_bool
9175     { \endpgfpicture }
9176     { \end { pgfpicture } }
9177     \fp_set:Nn \l_tmpa_fp
9178     {
9179       \fp_min:nn
9180       {
9181         \fp_min:nn
9182         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9183         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9184       }
9185       { 1.0 }
9186     }
9187     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9188     \pgfpicture
9189     \pgfrememberpicturepositiononpagetrue
9190     \pgf@relevantforpicturesizefalse
9191     \pgftransformshift
9192     {
9193       \pgfpoint
9194       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9195       { \dim_use:N \g_tmpa_dim }
9196     }

```



```

9197         \pgfnode
9198         { rectangle }
9199         { center }
9200         { \box_use:N \l_tmpa_box }
9201         { }
9202         { }
9203     \endpgfpicture
9204 }
9205 }
9206 }

```

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.

```

9207 \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.

```

9208 \bool_new:N \g_@@_footnote_bool

9209 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9210 {
9211     The~key~'\l_keys_key_str'~is~unknown. \\
9212     That~key~will~be~ignored. \\
9213     For~a~list~of~the~available~keys,~type~H~<return>.
9214 }
9215 {
9216     The~available~keys~are~(in~alphabetic~order):~
9217     footnote,~
9218     footnotehyper,~
9219     messages-for-Overleaf,~
9220     renew-dots,~and~
9221     renew-matrix.
9222 }

9223 \keys_define:nn { nicematrix / Package }
9224 {
9225     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9226     renew-dots .value_forbidden:n = true ,
9227     renew-matrix .code:n = \@@_renew_matrix: ,
9228     renew-matrix .value_forbidden:n = true ,
9229     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9230     footnote .bool_set:N = \g_@@_footnote_bool ,
9231     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.

```

9232     no-test-for-array .code:n = \prg_do_nothing: ,
9233     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9234 }
9235 \ProcessKeysOptions { nicematrix / Package }

```

```

9236 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9237 {
9238     You~can't~use~the~option~'footnote'~because~the~package~
9239     footnotehyper~has~already~been~loaded.~

```

```

9240     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9241     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9242     of~the~package~footnotehyper.\\
9243     The~package~footnote~won't~be~loaded.
9244 }
9245 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9246 {
9247     You~can't~use~the~option~'footnotehyper'~because~the~package~
9248     footnote~has~already~been~loaded.~
9249     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9250     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9251     of~the~package~footnote.\\
9252     The~package~footnotehyper~won't~be~loaded.
9253 }

9254 \bool_if:NT \g_@@_footnote_bool
9255 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9256     \IfClassLoadedTF { beamer }
9257     { \bool_set_false:N \g_@@_footnote_bool }
9258     {
9259         \IfPackageLoadedTF { footnotehyper }
9260         { \@@_error:n { footnote~with~footnotehyper~package } }
9261         { \usepackage { footnote } }
9262     }
9263 }

9264 \bool_if:NT \g_@@_footnotehyper_bool
9265 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9266     \IfClassLoadedTF { beamer }
9267     { \bool_set_false:N \g_@@_footnote_bool }
9268     {
9269         \IfPackageLoadedTF { footnote }
9270         { \@@_error:n { footnotehyper~with~footnote~package } }
9271         { \usepackage { footnotehyper } }
9272     }
9273     \bool_set_true:N \g_@@_footnote_bool
9274 }

```

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.

```

9275 \bool_new:N \l_@@_underscore_loaded_bool
9276 \IfPackageLoadedT { underscore }
9277 { \bool_set_true:N \l_@@_underscore_loaded_bool }

9278 \hook_gput_code:nnn { begindocument } { . }
9279 {
9280     \bool_if:NF \l_@@_underscore_loaded_bool
9281     {
9282         \IfPackageLoadedT { underscore }

```

```

9283         { \@@_error:n { underscore-after-nicematrix } }
9284     }
9285 }

```

39 Error messages of the package

```

9286 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9287 { \str_const:Nn \c_@@_available_keys_str { } }
9288 {
9289     \str_const:Nn \c_@@_available_keys_str
9290     { For-a-list-of-the-available-keys,-type-H<return>. }
9291 }
9292 \seq_new:N \g_@@_types_of_matrix_seq
9293 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9294 {
9295     NiceMatrix ,
9296     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9297 }
9298 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9299 { \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`.

```

9300 \cs_new_protected:Npn \@@_error_too_much_cols:
9301 {
9302     \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9303     { \@@_fatal:nn { too-much-cols-for-array } }
9304     \int_compare:nNnT \l_@@_last_col_int = { -2 }
9305     { \@@_fatal:n { too-much-cols-for-matrix } }
9306     \int_compare:nNnT \l_@@_last_col_int = { -1 }
9307     { \@@_fatal:n { too-much-cols-for-matrix } }
9308     \bool_if:NF \l_@@_last_col_without_value_bool
9309     { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9310 }

```

The following command must *not* be protected since it's used in an error message.

```

9311 \cs_new:Npn \@@_message_hdotsfor:
9312 {
9313     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9314     { ~Maybe-your-use-of-\token_to_str:N \Hdotsfor\ is-incorrect.}
9315 }
9316 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9317 {
9318     Incompatible-options.\
9319     You-should-not-use-'hvlines',~'rounded-corners'~and-'corners'~at-this-time.\
9320     The-output-will-not-be-reliable.
9321 }
9322 \@@_msg_new:nn { negative-weight }
9323 {
9324     Negative-weight.\
9325     The-weight-of-the-'X'-columns-must-be-positive-and-you-have-used-
9326     the-value~'\int_use:N \l_@@_weight_int'~.\
9327     The-absolute-value-will-be-used.
9328 }
9329 \@@_msg_new:nn { last-col-not-used }

```

```

9330 {
9331   Column~not~used.\\
9332   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9333   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9334 }
9335 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9336 {
9337   Too~much~columns.\\
9338   In~the~row~\int_eval:n { \c@iRow },~
9339   you~try~to~use~more~columns~
9340   than~allowed~by~your~\@@_full_name_env:. \@@_message_hdotsfor:\
9341   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9342   (plus~the~exterior~columns).~This~error~is~fatal.
9343 }
9344 \@@_msg_new:nn { too~much~cols~for~matrix }
9345 {
9346   Too~much~columns.\\
9347   In~the~row~\int_eval:n { \c@iRow },~
9348   you~try~to~use~more~columns~than~allowed~by~your~
9349   \@@_full_name_env:. \@@_message_hdotsfor:\ Recall~that~the~maximal~
9350   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9351   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9352   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9353   \token_to_str:N \setcounter\ to~change~that~value).~
9354   This~error~is~fatal.
9355 }
9356 \@@_msg_new:nn { too~much~cols~for~array }
9357 {
9358   Too~much~columns.\\
9359   In~the~row~\int_eval:n { \c@iRow },~
9360   ~you~try~to~use~more~columns~than~allowed~by~your~
9361   \@@_full_name_env:. \@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9362   \int_use:N \g_@@_static_num_of_col_int\
9363   ~(plus~the~potential~exterior~ones).~
9364   This~error~is~fatal.
9365 }
9366 \@@_msg_new:nn { columns~not~used }
9367 {
9368   Columns~not~used.\\
9369   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9370   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9371   The~columns~you~did~not~used~won't~be~created.\\
9372   You~won't~have~similar~error~message~till~the~end~of~the~document.
9373 }
9374 \@@_msg_new:nn { empty~preamble }
9375 {
9376   Empty~preamble.\\
9377   The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9378   This~error~is~fatal.
9379 }
9380 \@@_msg_new:nn { in~first~col }
9381 {
9382   Erroneous~use.\\
9383   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9384   That~command~will~be~ignored.
9385 }
9386 \@@_msg_new:nn { in~last~col }
9387 {
9388   Erroneous~use.\\
9389   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\

```

```

9390     That~command~will~be~ignored.
9391 }
9392 \@@_msg_new:nn { in~first~row }
9393 {
9394     Erroneous~use.\\
9395     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9396     That~command~will~be~ignored.
9397 }
9398 \@@_msg_new:nn { in~last~row }
9399 {
9400     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9401     That~command~will~be~ignored.
9402 }
9403 \@@_msg_new:nn { caption~outside~float }
9404 {
9405     Key~caption~forbidden.\\
9406     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9407     environment.~This~key~will~be~ignored.
9408 }
9409 \@@_msg_new:nn { short~caption~without~caption }
9410 {
9411     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9412     However,~your~'short~caption'~will~be~used~as~'caption'.
9413 }
9414 \@@_msg_new:nn { double~closing~delimiter }
9415 {
9416     Double~delimiter.\\
9417     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9418     delimiter.~This~delimiter~will~be~ignored.
9419 }
9420 \@@_msg_new:nn { delimiter~after~opening }
9421 {
9422     Double~delimiter.\\
9423     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9424     delimiter.~That~delimiter~will~be~ignored.
9425 }
9426 \@@_msg_new:nn { bad~option~for~line~style }
9427 {
9428     Bad~line~style.\\
9429     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9430     is~'standard'.~That~key~will~be~ignored.
9431 }
9432 \@@_msg_new:nn { Identical~notes~in~caption }
9433 {
9434     Identical~tabular~notes.\\
9435     You~can't~put~several~notes~with~the~same~content~in~
9436     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9437     If~you~go~on,~the~output~will~probably~be~erroneous.
9438 }
9439 \@@_msg_new:nn { tabularnote~below~the~tabular }
9440 {
9441     \token_to_str:N \tabularnote\ forbidden\\
9442     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9443     of~your~tabular~because~the~caption~will~be~composed~below~
9444     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9445     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9446     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9447     no~similar~error~will~raised~in~this~document.
9448 }

```

```

9449 \@@_msg_new:nn { Unknown~key~for~rules }
9450 {
9451   Unknown~key.\\
9452   There~is~only~two~keys~available~here:~width~and~color.\\
9453   Your~key~'\l_keys_key_str'~will~be~ignored.
9454 }
9455 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9456 {
9457   Unknown~key.\\
9458   There~is~only~two~keys~available~here:~
9459   'empty'~and~'not-empty'.\\
9460   Your~key~'\l_keys_key_str'~will~be~ignored.
9461 }
9462 \@@_msg_new:nn { Unknown~key~for~rotate }
9463 {
9464   Unknown~key.\\
9465   The~only~key~available~here~is~'c'.\\
9466   Your~key~'\l_keys_key_str'~will~be~ignored.
9467 }
9468 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9469 {
9470   Unknown~key.\\
9471   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9472   It~you~go~on,~you~will~probably~have~other~errors. \\
9473   \c_@@_available_keys_str
9474 }
9475 {
9476   The~available~keys~are~(in~alphabetic~order):~
9477   ccommand,~
9478   color,~
9479   command,~
9480   dotted,~
9481   letter,~
9482   multiplicity,~
9483   sep-color,~
9484   tikz,~and~total-width.
9485 }
9486 \@@_msg_new:nnn { Unknown~key~for~xdots }
9487 {
9488   Unknown~key.\\
9489   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9490   \c_@@_available_keys_str
9491 }
9492 {
9493   The~available~keys~are~(in~alphabetic~order):~
9494   'color',~
9495   'horizontal-labels',~
9496   'inter',~
9497   'line-style',~
9498   'radius',~
9499   'shorten',~
9500   'shorten-end'~and~'shorten-start'.
9501 }
9502 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9503 {
9504   Unknown~key.\\
9505   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9506   (and~you~try~to~use~'\l_keys_key_str')\\
9507   That~key~will~be~ignored.
9508 }
9509 \@@_msg_new:nn { label~without~caption }

```

```

9510 {
9511   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9512   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9513 }
9514 \@@_msg_new:nn { W~warning }
9515 {
9516   Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9517   (row~\int_use:N \c@iRow).
9518 }
9519 \@@_msg_new:nn { Construct~too~large }
9520 {
9521   Construct~too~large.\\
9522   Your~command~\token_to_str:N #1
9523   can't~be~drawn~because~your~matrix~is~too~small.\\
9524   That~command~will~be~ignored.
9525 }
9526 \@@_msg_new:nn { underscore~after~nicematrix }
9527 {
9528   Problem~with~'underscore'.\\
9529   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9530   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9531   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9532 }
9533 \@@_msg_new:nn { ampersand~in~light~syntax }
9534 {
9535   Ampersand~forbidden.\\
9536   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9537   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9538 }
9539 \@@_msg_new:nn { double~backslash~in~light~syntax }
9540 {
9541   Double~backslash~forbidden.\\
9542   You~can't~use~\token_to_str:N
9543   \\~to~separate~rows~because~the~key~'light-syntax'~
9544   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9545   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9546 }
9547 \@@_msg_new:nn { hlines~with~color }
9548 {
9549   Incompatible~keys.\\
9550   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9551   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9552   However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9553   Your~key~will~be~discarded.
9554 }
9555 \@@_msg_new:nn { bad~value~for~baseline }
9556 {
9557   Bad~value~for~baseline.\\
9558   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9559   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9560   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9561   the~form~'line-i'.\\
9562   A~value~of~1~will~be~used.
9563 }
9564 \@@_msg_new:nn { detection~of~empty~cells }
9565 {
9566   Problem~with~'not-empty'\\
9567   For~technical~reasons,~you~must~activate~
9568   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9569   in~order~to~use~the~key~'\l_keys_key_str'.\\

```

```

9570     That~key~will~be~ignored.
9571 }

9572 \@@_msg_new:nn { siunitx~not~loaded }
9573 {
9574     siunitx~not~loaded\\
9575     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9576     That~error~is~fatal.
9577 }

9578 \@@_msg_new:nn { Invalid~name }
9579 {
9580     Invalid~name.\\
9581     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9582     \SubMatrix\ of~your~\@@_full_name_env:~\\
9583     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9584     This~key~will~be~ignored.
9585 }

9586 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9587 {
9588     Wrong~line.\\
9589     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9590     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9591     number~is~not~valid.~It~will~be~ignored.
9592 }

9593 \@@_msg_new:nn { Impossible~delimiter }
9594 {
9595     Impossible~delimiter.\\
9596     It's~impossible~to~draw~the~#1~delimiter~of~your~
9597     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9598     in~that~column.
9599     \bool_if:NT \l_@@_submatrix_slim_bool
9600     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9601     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9602 }

9603 \@@_msg_new:nnn { width~without~X~columns }
9604 {
9605     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9606     That~key~will~be~ignored.
9607 }
9608 {
9609     This~message~is~the~message~'width~without~X~columns'~
9610     of~the~module~'nicematrix'.~
9611     The~experimented~users~can~disable~that~message~with~
9612     \token_to_str:N \msg_redirect_name:nnn.\\
9613 }
9614

9615 \@@_msg_new:nn { key~multiplicity~with~dotted }
9616 {
9617     Incompatible~keys. \\
9618     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9619     in~a~'custom~line'.~They~are~incompatible. \\
9620     The~key~'multiplicity'~will~be~discarded.
9621 }

9622 \@@_msg_new:nn { empty~environment }
9623 {
9624     Empty~environment.\\
9625     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9626 }

9627 \@@_msg_new:nn { No~letter~and~no~command }
9628 {
9629     Erroneous~use.\\

```



```

9630     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9631     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9632     ~'ccommand'~(to~draw~horizontal~rules).\\
9633     However,~you~can~go~on.
9634 }
9635 \@@_msg_new:nn { Forbidden~letter }
9636 {
9637     Forbidden~letter.\\
9638     You-can't-use-the-letter~'#1'~for~a~customized-line.\\
9639     It~will~be~ignored.
9640 }
9641 \@@_msg_new:nn { Several~letters }
9642 {
9643     Wrong~name.\\
9644     You-must-use-only-one-letter~as~value~for~the~key~'letter'~(and~you~
9645     have-used~'\l_@@_letter_str').\\
9646     It~will~be~ignored.
9647 }
9648 \@@_msg_new:nn { Delimiter-with-small }
9649 {
9650     Delimiter~forbidden.\\
9651     You-can't-put-a-delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
9652     because~the~key~'small'~is~in~force.\\
9653     This~error~is~fatal.
9654 }
9655 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9656 {
9657     Unknown~cell.\\
9658     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9659     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
9660     can't~be~executed~because~a~cell~doesn't~exist.\\
9661     This~command~\token_to_str:N \line\ will~be~ignored.
9662 }
9663 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9664 {
9665     Duplicate~name.\\
9666     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
9667     in~this~\@@_full_name_env:.\\
9668     This~key~will~be~ignored.\\
9669     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9670     { For~a~list~of~the~names~already~used,~type~H<return>. }
9671 }
9672 {
9673     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9674     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9675 }
9676 \@@_msg_new:nn { r~or~l~with~preamble }
9677 {
9678     Erroneous~use.\\
9679     You-can't-use-the-key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9680     You-must-specify~the~alignment~of~your~columns~with~the~preamble~of~
9681     your~\@@_full_name_env:.\\
9682     This~key~will~be~ignored.
9683 }
9684 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9685 {
9686     Erroneous~use.\\
9687     You-can't-use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9688     the~array.~This~error~is~fatal.
9689 }

```

```

9690 \@@_msg_new:nn { bad-corner }
9691 {
9692   Bad-corner.\
9693   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
9694   'corners').~The-available-values-are:~NW,~SW,~NE~and~SE.\
9695   This-specification-of~corner~will-be-ignored.
9696 }
9697 \@@_msg_new:nn { bad-border }
9698 {
9699   Bad-border.\
9700   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9701   (in-the-key~'borders'~of~the-command~\token_to_str:N \Block).~
9702   The-available-values-are:~left,~right,~top~and~bottom~(and-you-can~
9703   also~use~the-key~'tikz'
9704   \IfPackageLoadedF { tikz }
9705     {~if-you-load-the-LaTeX-package~'tikz'}).\\
9706   This-specification-of~border~will-be-ignored.
9707 }
9708 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9709 {
9710   TikZ-not-loaded.\
9711   You-can't-use~\token_to_str:N \TikzEveryCell\
9712   because-you-have-not-loaded-tikz.~
9713   This-command-will-be-ignored.
9714 }
9715 \@@_msg_new:nn { tikz-key~without~tikz }
9716 {
9717   TikZ-not-loaded.\
9718   You-can't-use~the-key~'tikz'~for~the-command~'\token_to_str:N
9719   \Block'~because-you-have-not-loaded-tikz.~
9720   This-key-will-be-ignored.
9721 }
9722 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
9723 {
9724   Erroneous-use.\
9725   In-the~\@@_full_name_env:,~you-must-use~the-key~
9726   'last-col'~without~value.\
9727   However,~you-can-go-on-for~this-time~
9728   (the-value~'\l_keys_value_tl'~will-be-ignored).
9729 }
9730 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9731 {
9732   Erroneous-use.\
9733   In~\token_to_str:N \NiceMatrixOptions,~you-must-use~the-key~
9734   'last-col'~without~value.\
9735   However,~you-can-go-on-for~this-time~
9736   (the-value~'\l_keys_value_tl'~will-be-ignored).
9737 }
9738 \@@_msg_new:nn { Block~too~large~1 }
9739 {
9740   Block~too~large.\
9741   You-try-to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9742   too~small~for~that~block. \
9743   This~block~and~maybe~others~will-be-ignored.
9744 }
9745 \@@_msg_new:nn { Block~too~large~2 }
9746 {
9747   Block~too~large.\
9748   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9749   \g_@@_static_num_of_col_int\
9750   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~

```

```

9751     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9752     (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:~\\
9753     This-block-and-maybe-others-will-be-ignored.
9754 }

9755 \@@_msg_new:nn { unknown~column-type }
9756 {
9757     Bad-column-type.\\
9758     The-column-type-#1'~in-your~\@@_full_name_env:\
9759     is-unknown. \\
9760     This-error-is-fatal.
9761 }

9762 \@@_msg_new:nn { unknown~column-type-S }
9763 {
9764     Bad-column-type.\\
9765     The-column-type-'S'~in-your~\@@_full_name_env:\ is-unknown. \\
9766     If-you-want-to-use-the-column-type-'S'~of-siunitx,~you-should~
9767     load-that-package. \\
9768     This-error-is-fatal.
9769 }

9770 \@@_msg_new:nn { tabularnote~forbidden }
9771 {
9772     Forbidden-command.\\
9773     You-can't-use-the-command~\token_to_str:N\tabularnote\
9774     ~here.~This-command-is-available-only-in~
9775     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9776     the-argument-of-a-command~\token_to_str:N \caption\ included~
9777     in-an-environment~{table}. \\
9778     This-command-will-be-ignored.
9779 }

9780 \@@_msg_new:nn { borders~forbidden }
9781 {
9782     Forbidden-key.\\
9783     You-can't-use-the-key~'borders'~of-the-command~\token_to_str:N \Block\
9784     because-the-option~'rounded-corners'~
9785     is-in-force-with-a-non-zero-value.\\
9786     This-key-will-be-ignored.
9787 }

9788 \@@_msg_new:nn { bottomrule~without~booktabs }
9789 {
9790     booktabs-not-loaded.\\
9791     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
9792     loaded~'booktabs'.\\
9793     This-key-will-be-ignored.
9794 }

9795 \@@_msg_new:nn { enumitem~not~loaded }
9796 {
9797     enumitem-not-loaded.\\
9798     You-can't-use-the-command~\token_to_str:N\tabularnote\
9799     ~because-you-haven't~loaded~'enumitem'.\\
9800     All-the-commands~\token_to_str:N\tabularnote\ will-be~
9801     ignored-in-the-document.
9802 }

9803 \@@_msg_new:nn { tikz~without~tikz }
9804 {
9805     Tikz-not-loaded.\\
9806     You-can't-use-the-key~'tikz'~here~because~Tikz-is-not~
9807     loaded.~If-you-go-on,~that-key-will-be-ignored.
9808 }

9809 \@@_msg_new:nn { tikz-in~custom~line~without~tikz }
9810 {

```

```

9811 Tikz~not~loaded.\\
9812 You~have~used~the~key~'tikz'~in~the~definition~of~a~
9813 customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9814 You~can~go~on~but~you~will~have~another~error~if~you~actually~
9815 use~that~custom~line.
9816 }
9817 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9818 {
9819 Tikz~not~loaded.\\
9820 You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9821 command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9822 That~key~will~be~ignored.
9823 }
9824 \@@_msg_new:nn { without~color~inside }
9825 {
9826 If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9827 \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9828 outside~\token_to_str:N \CodeBefore,~you~
9829 should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\
9830 You~can~go~on~but~you~may~need~more~compilations.
9831 }
9832 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9833 {
9834 Erroneous~use.\\
9835 In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9836 which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9837 The~key~'color'~will~be~discarded.
9838 }
9839 \@@_msg_new:nn { Wrong~last~row }
9840 {
9841 Wrong~number.\\
9842 You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9843 \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9844 If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9845 last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9846 without~value~(more~compilations~might~be~necessary).
9847 }
9848 \@@_msg_new:nn { Yet~in~env }
9849 {
9850 Nested~environments.\\
9851 Environments~of~nicematrix~can't~be~nested.\\
9852 This~error~is~fatal.
9853 }
9854 \@@_msg_new:nn { Outside~math~mode }
9855 {
9856 Outside~math~mode.\\
9857 The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9858 (and~not~in~\token_to_str:N \vcenter).\\
9859 This~error~is~fatal.
9860 }
9861 \@@_msg_new:nn { One~letter~allowed }
9862 {
9863 Bad~name.\\
9864 The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9865 It~will~be~ignored.
9866 }
9867 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9868 {
9869 Environment~{TabularNote}~forbidden.\\
9870 You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~

```

```

9871     but~*before*~the~\token_to_str:N \CodeAfter.\\
9872     This~environment~{TabularNote}~will~be~ignored.
9873 }
9874 \@@_msg_new:nn { varwidth~not~loaded }
9875 {
9876     varwidth~not~loaded.\\
9877     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9878     loaded.\\
9879     Your~column~will~behave~like~'p'.
9880 }
9881 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9882 {
9883     Unknow~key.\\
9884     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9885     \c_@@_available_keys_str
9886 }
9887 {
9888     The~available~keys~are~(in~alphabetic~order):~
9889     color,~
9890     dotted,~
9891     multiplicity,~
9892     sep~color,~
9893     tikz,~and~total~width.
9894 }
9895
9896 \@@_msg_new:nnn { Unknown~key~for~Block }
9897 {
9898     Unknown~key.\\
9899     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9900     \Block.\\ It~will~be~ignored. \\
9901     \c_@@_available_keys_str
9902 }
9903 {
9904     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
9905     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
9906     opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
9907     and~vlines.
9908 }
9909 \@@_msg_new:nnn { Unknown~key~for~Brace }
9910 {
9911     Unknown~key.\\
9912     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9913     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9914     It~will~be~ignored. \\
9915     \c_@@_available_keys_str
9916 }
9917 {
9918     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9919     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9920     right~shorten)~and~yshift.
9921 }
9922 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9923 {
9924     Unknown~key.\\
9925     The~key~'\l_keys_key_str'~is~unknown.\\
9926     It~will~be~ignored. \\
9927     \c_@@_available_keys_str
9928 }
9929 {
9930     The~available~keys~are~(in~alphabetic~order):~
9931     delimiters/color,~
9932     rules~(with~the~subkeys~'color'~and~'width'),~

```

```

9933     sub-matrix~(several~subkeys)~
9934     and~xdots~(several~subkeys).~
9935     The~latter~is~for~the~command~\token_to_str:N \line.
9936 }

9937 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9938 {
9939     Unknown~key.\\
9940     The~key~'\l_keys_key_str'~is~unknown.\\
9941     It~will~be~ignored. \\
9942     \c_@@_available_keys_str
9943 }
9944 {
9945     The~available~keys~are~(in~alphabetic~order):~
9946     create~cell~nodes,~
9947     delimiters/color~and~
9948     sub-matrix~(several~subkeys).
9949 }

9950 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9951 {
9952     Unknown~key.\\
9953     The~key~'\l_keys_key_str'~is~unknown.\\
9954     That~key~will~be~ignored. \\
9955     \c_@@_available_keys_str
9956 }
9957 {
9958     The~available~keys~are~(in~alphabetic~order):~
9959     'delimiters/color',~
9960     'extra-height',~
9961     'hlines',~
9962     'hvlines',~
9963     'left-xshift',~
9964     'name',~
9965     'right-xshift',~
9966     'rules'~(with~the~subkeys~'color'~and~'width'),~
9967     'slim',~
9968     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9969     and~'right-xshift').\\
9970 }

9971 \@@_msg_new:nnn { Unknown~key~for~notes }
9972 {
9973     Unknown~key.\\
9974     The~key~'\l_keys_key_str'~is~unknown.\\
9975     That~key~will~be~ignored. \\
9976     \c_@@_available_keys_str
9977 }
9978 {
9979     The~available~keys~are~(in~alphabetic~order):~
9980     bottomrule,~
9981     code~after,~
9982     code~before,~
9983     detect~duplicates,~
9984     enumitem~keys,~
9985     enumitem~keys~para,~
9986     para,~
9987     label~in~list,~
9988     label~in~tabular~and~
9989     style.
9990 }

9991 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9992 {
9993     Unknown~key.\\
9994     The~key~'\l_keys_key_str'~is~unknown~for~the~command~

```

```

9995     \token_to_str:N \RowStyle. \\
9996     That~key~will~be~ignored. \\
9997     \c_@@_available_keys_str
9998 }
9999 {
10000     The~available~keys~are~(in~alphabetic~order):~
10001     'bold',~
10002     'cell-space-top-limit',~
10003     'cell-space-bottom-limit',~
10004     'cell-space-limits',~
10005     'color',~
10006     'nb-rows'~and~
10007     'rowcolor'.
10008 }
10009 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10010 {
10011     Unknown~key.\\
10012     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10013     \token_to_str:N \NiceMatrixOptions. \\
10014     That~key~will~be~ignored. \\
10015     \c_@@_available_keys_str
10016 }
10017 {
10018     The~available~keys~are~(in~alphabetic~order):~
10019     &~in~blocks,~
10020     allow~duplicate~names,~
10021     ampersand~in~blocks,~
10022     caption~above,~
10023     cell-space-bottom-limit,~
10024     cell-space-limits,~
10025     cell-space-top-limit,~
10026     code~for~first~col,~
10027     code~for~first~row,~
10028     code~for~last~col,~
10029     code~for~last~row,~
10030     corners,~
10031     custom~key,~
10032     create~extra~nodes,~
10033     create~medium~nodes,~
10034     create~large~nodes,~
10035     custom~line,~
10036     delimiters~(several~subkeys),~
10037     end~of~row,~
10038     first~col,~
10039     first~row,~
10040     hlines,~
10041     hvlines,~
10042     hvlines~except~borders,~
10043     last~col,~
10044     last~row,~
10045     left~margin,~
10046     light~syntax,~
10047     light~syntax~expanded,~
10048     matrix/columns~type,~
10049     no~cell~nodes,~
10050     notes~(several~subkeys),~
10051     nullify~dots,~
10052     pgf~node~code,~
10053     renew~dots,~
10054     renew~matrix,~
10055     respect~arraystretch,~
10056     rounded~corners,~
10057     right~margin,~

```

```

10058     rules~(with~the~subkeys~'color'~and~'width'),~
10059     small,~
10060     sub-matrix~(several~subkeys),~
10061     vlines,~
10062     xdots~(several~subkeys).
10063 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10064 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10065 {
10066     Unknown~key.\\
10067     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10068     \{NiceArray\}. \\
10069     That~key~will~be~ignored. \\
10070     \c_@@_available_keys_str
10071 }
10072 {
10073     The~available~keys~are~(in~alphabetic~order):~
10074     &~in~blocks,~
10075     ampersand~in~blocks,~
10076     b,~
10077     baseline,~
10078     c,~
10079     cell-space-bottom-limit,~
10080     cell-space-limits,~
10081     cell-space-top-limit,~
10082     code~after,~
10083     code~for~first~col,~
10084     code~for~first~row,~
10085     code~for~last~col,~
10086     code~for~last~row,~
10087     color~inside,~
10088     columns~width,~
10089     corners,~
10090     create~extra~nodes,~
10091     create~medium~nodes,~
10092     create~large~nodes,~
10093     extra~left~margin,~
10094     extra~right~margin,~
10095     first~col,~
10096     first~row,~
10097     hlines,~
10098     hvlines,~
10099     hvlines~except~borders,~
10100     last~col,~
10101     last~row,~
10102     left~margin,~
10103     light~syntax,~
10104     light~syntax~expanded,~
10105     name,~
10106     no~cell~nodes,~
10107     nullify~dots,~
10108     pgf~node~code,~
10109     renew~dots,~
10110     respect~arraystretch,~
10111     right~margin,~
10112     rounded~corners,~
10113     rules~(with~the~subkeys~'color'~and~'width'),~
10114     small,~
10115     t,~
10116     vlines,~
10117     xdots/color,~
10118     xdots/shorten~start,~

```



```

10119     xdots/shorten-end,~
10120     xdots/shorten-and~
10121     xdots/line-style.
10122 }

```

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`).

```

10123 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10124 {
10125   Unknown~key.\\
10126   The~key~'\l_keys_key_str'~is~unknown~for~the~
10127   \@@_full_name_env:. \\
10128   That~key~will~be~ignored. \\
10129   \c_@@_available_keys_str
10130 }
10131 {
10132   The~available~keys~are~(in~alphabetic~order):~
10133   &-in-blocks,~
10134   ampersand-in-blocks,~
10135   b,~
10136   baseline,~
10137   c,~
10138   cell-space-bottom-limit,~
10139   cell-space-limits,~
10140   cell-space-top-limit,~
10141   code-after,~
10142   code-for-first-col,~
10143   code-for-first-row,~
10144   code-for-last-col,~
10145   code-for-last-row,~
10146   color-inside,~
10147   columns-type,~
10148   columns-width,~
10149   corners,~
10150   create-extra-nodes,~
10151   create-medium-nodes,~
10152   create-large-nodes,~
10153   extra-left-margin,~
10154   extra-right-margin,~
10155   first-col,~
10156   first-row,~
10157   hlines,~
10158   hvlines,~
10159   hvlines-except-borders,~
10160   l,~
10161   last-col,~
10162   last-row,~
10163   left-margin,~
10164   light-syntax,~
10165   light-syntax-expanded,~
10166   name,~
10167   no-cell-nodes,~
10168   nullify-dots,~
10169   pgf-node-code,~
10170   r,~
10171   renew-dots,~
10172   respect-arraystretch,~
10173   right-margin,~
10174   rounded-corners,~
10175   rules~(with~the~subkeys~'color'~and~'width'),~
10176   small,~
10177   t,~
10178   vlides,~
10179   xdots/color,~

```

```

10180     xdots/shorten-start,~
10181     xdots/shorten-end,~
10182     xdots/shorten-and~
10183     xdots/line-style.
10184 }
10185 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10186 {
10187     Unknown~key.\\
10188     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10189     \{NiceTabular\}. \\
10190     That~key~will~be~ignored. \\
10191     \c_@@_available_keys_str
10192 }
10193 {
10194     The~available~keys~are~(in~alphabetic~order):~
10195     &~in~blocks,~
10196     ampersand~in~blocks,~
10197     b,~
10198     baseline,~
10199     c,~
10200     caption,~
10201     cell-space-bottom-limit,~
10202     cell-space-limits,~
10203     cell-space-top-limit,~
10204     code-after,~
10205     code-for-first-col,~
10206     code-for-first-row,~
10207     code-for-last-col,~
10208     code-for-last-row,~
10209     color-inside,~
10210     columns-width,~
10211     corners,~
10212     custom-line,~
10213     create-extra-nodes,~
10214     create-medium-nodes,~
10215     create-large-nodes,~
10216     extra-left-margin,~
10217     extra-right-margin,~
10218     first-col,~
10219     first-row,~
10220     hlines,~
10221     hvlines,~
10222     hvlines-except-borders,~
10223     label,~
10224     last-col,~
10225     last-row,~
10226     left-margin,~
10227     light-syntax,~
10228     light-syntax-expanded,~
10229     name,~
10230     no-cell-nodes,~
10231     notes~(several~subkeys),~
10232     nullify-dots,~
10233     pgf-node-code,~
10234     renew-dots,~
10235     respect-arraystretch,~
10236     right-margin,~
10237     rounded-corners,~
10238     rules~(with~the~subkeys~'color'~and~'width'),~
10239     short-caption,~
10240     t,~
10241     tabularnote,~
10242     vlides,~

```

```

10243     xdots/color,~
10244     xdots/shorten-start,~
10245     xdots/shorten-end,~
10246     xdots/shorten-and~
10247     xdots/line-style.
10248 }
10249 \@@_msg_new:nnn { Duplicate-name }
10250 {
10251   Duplicate~name.\\
10252   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10253   the~same~environment~name~twice.~You~can~go~on,~but,~
10254   maybe,~you~will~have~incorrect~results~especially~
10255   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10256   message~again,~use~the~key~'allow-duplicate-names'~in~
10257   '\token_to_str:N \NiceMatrixOptions'.\\
10258   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10259     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10260 }
10261 {
10262   The~names~already~defined~in~this~document~are:~
10263   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10264 }
10265 \@@_msg_new:nn { Option~auto~for~columns~width }
10266 {
10267   Erroneous~use.\\
10268   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10269   That~key~will~be~ignored.
10270 }
10271 \@@_msg_new:nn { NiceTabularX~without~X }
10272 {
10273   NiceTabularX~without~X.\\
10274   You~should~not~use~{NiceTabularX}~without~X~columns.\\
10275   However,~you~can~go~on.
10276 }
10277 \@@_msg_new:nn { Preamble~forgotten }
10278 {
10279   Preamble~forgotten.\\
10280   You~have~probably~forgotten~the~preamble~of~your~
10281   \@@_full_name_env:. \\
10282   This~error~is~fatal.
10283 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	We construct the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	91
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
15	After the construction of the array	93
16	We draw the dotted lines	100
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	114
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	142
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	159
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	189
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

31	The keyword <code>\CodeAfter</code>	192
32	The delimiters in the preamble	193
33	The command <code>\SubMatrix</code>	194
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	202
35	The command <code>TikzEveryCell</code>	205
36	The command <code>\ShowCellNames</code>	207
37	We process the options at package loading	209
38	About the package underscore	211
39	Error messages of the package	211