

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

January 12, 2025

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 7.0x of `nicematrix`, at the date of 2025/01/07.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

97 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

```

208     \skip_horizontal:N \c_zero_dim
209   }

```

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

214 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

278 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
332 {
```

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

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

```
335   \fi:
```

```
336 }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

385 \bool_new:N \g_@@_empty_cell_bool

```

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

```

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

```

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

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

```

388 \seq_new:N \g_@@_blocks_seq

```

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

```

389 \seq_new:N \g_@@_pos_of_blocks_seq

```

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

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

```

390 \seq_new:N \g_@@_future_pos_of_blocks_seq

```

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

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

```

391 \seq_new:N \g_@@_pos_of_xdots_seq

```

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

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

```

392 \seq_new:N \g_@@_pos_of_stroken_blocks_seq

```

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

```

393 \clist_new:N \l_@@_corners_cells_clist

```

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

```

394 \seq_new:N \g_@@_submatrix_names_seq

```

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

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

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

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

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

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

The following counters will be used when drawing the rules.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


Variables for the exterior rows and columns

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

- **First row**

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

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

- **First column**

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

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

- **Last row**

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

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

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

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

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

```

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

```

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

```

449 \bool_new:N \g_@@_last_col_found_bool

```

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

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

```

450 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

451 \cs_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 that `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```

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

```

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

```

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

```

The following internal parameters are for:

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

```

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

```

5 The command `\tabularnote`

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

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

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

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

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

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

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

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

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

```
487 \seq_new:N \l_@@_notes_labels_seq
488 \newcounter { nicematrix_draft }
```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

{label}{text of the tabularnote}.

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

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

```

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

```

```

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

```

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

```

589     \hbox_set:Nn \l_tmpa_box
590     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

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

```

672 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
673 {
674   \begin { pgfscope }
675   \pgfset
676   {
677     inner~sep = \c_zero_dim ,
678     minimum~size = \c_zero_dim
679   }
680   \pgftransformshift { \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 behaviour of `\cline` is changed in the environments of `nicematrix`: a `\cline` spreads the array by an amount equal to `\arrayrulewidth`. It's possible to disable this feature with the key `\l_@@_standard_line_bool`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

989 \keys_define:nn { nicematrix }
990 {

```

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8 Important code used by {NiceArrayWithDelims}

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

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

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

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

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

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

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

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

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

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

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

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

The following command is nullified in the tabulars.

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

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

```

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

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

The following commands are nullified in the tabulars.

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

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

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1289     \@@_update_max_cell_width:

```

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

```

1290     \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```


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

```

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

```

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

```

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

```

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

For example, for the following matrix,

```

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

```

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

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).

```

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

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

```

```

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

```

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

```

1422 \@tabarray

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` before the vertical skip (and thus, at a wrong place). That's 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 redefinition⁴.

```

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

```

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

```

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

1535 \cs_new_protected:Npn \@_pre_array_ii:
1536 {

```

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

```

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

```

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

```

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

```

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

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

We keep in memory the old versions 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).

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

1638 \int_gzero_new:N \g_@@_row_total_int

```

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

```

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

```

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

```

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

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

```

Now the `\CodeBefore.`

```

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

```

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

```

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

```

Idem for other sequences written on the aux file.

```

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

```

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

```

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

```

The value -2 is important.

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

```

1685   \@@_pre_array_ii:

```

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

```

1686   \box_clear_new:N \l_@@_the_array_box

```

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

```

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

```

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

```

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

```


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

```

1701 \hbox_set:Nw \l_@@_the_array_box
1702 \skip_horizontal:N \l_@@_left_margin_dim
1703 \skip_horizontal:N \l_@@_extra_left_margin_dim
1704 \bool_if:NT \c_@@_recent_array_bool
1705 { \UseTaggingSocket { tbl / hmode / begin } }

```

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

```

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

```

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

```

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

```

We go on with `\@@_pre_array:` which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```

1719 \@@_pre_array:
1720 }

```

9 The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed (that command will be used only once and is present alone only for legibility).

```

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

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1723 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1724 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1725 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1726 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

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

```

First, the recreation of the row nodes.

```

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

```

Now, the recreation of the col nodes.

```

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

```

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

```

1743 \@@_create_diag_nodes:

```

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

```

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

```

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

```

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

```

```

1769 \cs_new_protected:Npn \@@_exec_code_before:
1770 {

```

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...

```

1771 \clist_map_inline:Nn \l_@@_corners_cells_clist
1772 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1773 \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`.

```

1774 \@@_add_to_colors_seq:nn { { nocolor } } { }
1775 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1776 \group_begin:

```

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

```
1777 \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`).

```
1778 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1779 { \@@_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`.

```
1780 \exp_last_unbraced:No \@@_CodeBefore_keys:
1781 \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.

```
1782 \@@_actually_color:
1783 \l_@@_code_before_tl
1784 \q_stop
1785 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1786 \group_end:
1787 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1788 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1789 }
```

```
1790 \keys_define:nn { nicematrix / CodeBefore }
1791 {
1792   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1793   create-cell-nodes .default:n = true ,
1794   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1795   sub-matrix .value_required:n = true ,
1796   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1797   delimiters / color .value_required:n = true ,
1798   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1799 }
1800 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1801 {
1802   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1803   \@@_CodeBefore:w
1804 }
```

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.

```
1805 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1806 {
1807   \bool_if:NT \g_@@_aux_found_bool
1808   {
1809     \@@_pre_code_before:
1810     #1
1811   }
1812 }
```

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.

```

1813 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1814 {
1815   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1816   {
1817     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1818     \pgfcoordinate { \@@_env: - row - ##1 - base }
1819     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1820     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1821     {
1822       \cs_if_exist:cT
1823       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1824       {
1825         \pgfsys@getposition
1826         { \@@_env: - ##1 - #####1 - NW }
1827         \@@_node_position:
1828         \pgfsys@getposition
1829         { \@@_env: - ##1 - #####1 - SE }
1830         \@@_node_position_i:
1831         \@@_pgf_rect_node:nnn
1832         { \@@_env: - ##1 - #####1 }
1833         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1834         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1835       }
1836     }
1837   }
1838   \int_step_inline:nn \c@iRow
1839   {
1840     \pgfnodealias
1841     { \@@_env: - ##1 - last }
1842     { \@@_env: - ##1 - \int_use:N \c@jCol }
1843   }
1844   \int_step_inline:nn \c@jCol
1845   {
1846     \pgfnodealias
1847     { \@@_env: - last - ##1 }
1848     { \@@_env: - \int_use:N \c@iRow - ##1 }
1849   }
1850   \@@_create_extra_nodes:
1851 }

```

```

1852 \cs_new_protected:Npn \@@_create_blocks_nodes:
1853 {
1854   \pgfpicture
1855   \pgf@relevantforpicturesizefalse
1856   \pgfrememberpicturepositiononpagetrue
1857   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1858   { \@@_create_one_block_node:nnnnn ##1 }
1859   \endpgfpicture
1860 }

```

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

```

1861 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1862 {
1863   \tl_if_empty:nF { #5 }
1864   {
1865     \@@_qpoint:n { col - #2 }

```

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).

```

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

1881 \cs_new_protected:Npn \@@_patch_for_revtext:
1882 {
1883 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1884 \cs_set_eq:NN \@array \@array@array
1885 \cs_set_eq:NN \@tabular \@tabular@array
1886 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1887 \cs_set_eq:NN \array \array@array
1888 \cs_set_eq:NN \endarray \endarray@array
1889 \cs_set:Npn \endtabular { \endarray $\egroup } % $
1890 \cs_set_eq:NN \@mkpream \@mkpream@array
1891 \cs_set_eq:NN \@classx \@classx@array
1892 \cs_set_eq:NN \insert@column \insert@column@array
1893 \cs_set_eq:NN \@arraycr \@arraycr@array
1894 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1895 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1896 }

```

10 The environment `{NiceArrayWithDelims}`

```

1897 \NewDocumentEnvironment { NiceArrayWithDelims }
1898 { m m O { } m ! O { } t \CodeBefore }
1899 {
1900 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1901 \@@_provide_pgfsyspdfmark:
1902 \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.

```

1903 \bgroup

1904 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1905 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1906 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1907 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1908 \int_gzero:N \g_@@_block_box_int
1909 \dim_zero:N \g_@@_width_last_col_dim
1910 \dim_zero:N \g_@@_width_first_col_dim
1911 \bool_gset_false:N \g_@@_row_of_col_done_bool
1912 \str_if_empty:NT \g_@@_name_env_str
1913 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1914 \bool_if:NTF \l_@@_tabular_bool

```

```

1915 \mode_leave_vertical:
1916 \@@_test_if_math_mode:
1917 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1918 \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.

```

1919 \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.

```

1920 \cs_if_exist:NT \tikz@library@external@loaded
1921 {
1922   \tikzexternaldisable
1923   \cs_if_exist:NT \ifstandalone
1924     { \tikzset { external / optimize = false } }
1925 }

```

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

```

1926 \int_gincr:N \g_@@_env_int
1927 \bool_if:NF \l_@@_block_auto_columns_width_bool
1928 { \dim_gzero_new:N \g_@@_max_cell_width_dim }

```

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

```

1929 \seq_gclear:N \g_@@_blocks_seq
1930 \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`.

```

1931 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1932 \seq_gclear:N \g_@@_pos_of_xdots_seq
1933 \tl_gclear_new:N \g_@@_code_before_tl
1934 \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.

```

1935 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1936 {
1937   \bool_gset_true:N \g_@@_aux_found_bool
1938   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1939 }
1940 { \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.

```

1941 \tl_gclear:N \g_@@_aux_tl
1942 \tl_if_empty:NF \g_@@_code_before_tl
1943 {
1944   \bool_set_true:N \l_@@_code_before_bool
1945   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1946 }
1947 \tl_if_empty:NF \g_@@_pre_code_before_tl
1948 { \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`.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1949 \bool_if:NTF \g_@@_delims_bool
1950   { \keys_set:nn { nicematrix / pNiceArray } }
1951   { \keys_set:nn { nicematrix / NiceArray } }
1952   { #3 , #5 }

1953 \@@_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:.

```

1954 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1955 }

```

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

```

1956 {
1957   \bool_if:NTF \l_@@_light_syntax_bool
1958   { \use:c { end @@-light-syntax } }
1959   { \use:c { end @@-normal-syntax } }
1960   \c_math_toggle_token
1961   \skip_horizontal:N \l_@@_right_margin_dim
1962   \skip_horizontal:N \l_@@_extra_right_margin_dim
1963
1964   % awful workaround
1965   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1966   {
1967     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1968     {
1969       \skip_horizontal:N - \l_@@_columns_width_dim
1970       \bool_if:NTF \l_@@_tabular_bool
1971       { \skip_horizontal:n { - 2 \tabcolsep } }
1972       { \skip_horizontal:n { - 2 \arraycolsep } }
1973     }
1974   }
1975   \hbox_set_end:
1976   \bool_if:NT \c_@@_recent_array_bool
1977   { \UseTaggingSocket { tbl / hmode / end } }

```

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

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

```

1978 \bool_if:NT \l_@@_width_used_bool
1979   {
1980     \int_if_zero:nT \g_@@_total_X_weight_int
1981     { \@@_error_or_warning:n { width-without-X~columns } }
1982   }

```

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 .

```

1983 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1984   { \@@_compute_width_X: }

```

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

```

1985 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1986   {
1987     \bool_if:NF \l_@@_last_row_without_value_bool
1988     {
1989       \int_compare:nNnF \l_@@_last_row_int = \c_iRow

```

```

1990         {
1991             \@@_error:n { Wrong~last~row }
1992             \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1993         }
1994     }
1995 }

```

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

```

1996     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1997     \bool_if:NTF \g_@@_last_col_found_bool
1998     { \int_gdecr:N \c@jCol }
1999     {
2000         \int_compare:nNnT \l_@@_last_col_int > { -1 }
2001         { \@@_error:n { last~col~not~used } }
2002     }

```

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

```

2003     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2004     \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).

```

2005     \int_if_zero:nT \l_@@_first_col_int
2006     { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2007     \bool_if:nTF { ! \g_@@_delims_bool }
2008     {
2009         \str_if_eq:eeTF \l_@@_baseline_tl { c }
2010         \@@_use_arraybox_with_notes_c:
2011         {
2012             \str_if_eq:eeTF \l_@@_baseline_tl { b }
2013             \@@_use_arraybox_with_notes_b:
2014             \@@_use_arraybox_with_notes:
2015         }
2016     }

```

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).

```

2017     {
2018         \int_if_zero:nTF \l_@@_first_row_int
2019         {
2020             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2021             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2022         }
2023         { \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

```

2024     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2025     {
2026         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2027         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2028     }
2029     { \dim_zero:N \l_tmpb_dim }
2030     \hbox_set:Nn \l_tmpa_box
2031     {
2032         \m@th % added 2024/11/21

```

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

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


```

2033         \c_math_toggle_token
2034         \@@_color:o \l_@@_delimiters_color_tl
2035         \exp_after:wN \left \g_@@_left_delim_tl
2036         \vcenter
2037         {

```

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.

```

2038         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2039         \hbox
2040         {
2041             \bool_if:NTF \l_@@_tabular_bool
2042             { \skip_horizontal:N -\tabcolsep }
2043             { \skip_horizontal:N -\arraycolsep }
2044             \@@_use_arraybox_with_notes_c:
2045             \bool_if:NTF \l_@@_tabular_bool
2046             { \skip_horizontal:N -\tabcolsep }
2047             { \skip_horizontal:N -\arraycolsep }
2048         }

```

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

```

2049         \skip_vertical:N -\l_tmpb_dim
2050         \skip_vertical:N \arrayrulewidth
2051     }
2052     \exp_after:wN \right \g_@@_right_delim_tl
2053     \c_math_toggle_token
2054 }

```

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.

```

2055     \bool_if:NTF \l_@@_delimiters_max_width_bool
2056     {
2057         \@@_put_box_in_flow_bis:nn
2058         \g_@@_left_delim_tl
2059         \g_@@_right_delim_tl
2060     }
2061     \@@_put_box_in_flow:
2062 }

```

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).

```

2063     \bool_if:NT \g_@@_last_col_found_bool
2064     { \skip_horizontal:N \g_@@_width_last_col_dim }
2065     \bool_if:NT \l_@@_preamble_bool
2066     {
2067         \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2068         { \@@_warning_gredirect_none:n { columns-not-used } }
2069     }
2070     \@@_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.

```

2071     \egroup

```

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

```

2072     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2073     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2074     \iow_now:Ne \@mainaux
2075     {
2076         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2077         { \exp_not:o \g_@@_aux_tl }
2078     }
2079     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

```

```

2080 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2081 }

```

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

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

```

2082 \cs_new_protected:Npn \@@_compute_width_X:
2083 {
2084   \tl_gput_right:Ne \g_@@_aux_tl
2085   {
2086     \bool_set_true:N \l_@@_X_columns_aux_bool
2087     \dim_set:Nn \l_@@_X_columns_dim
2088     {
2089       \dim_compare:nNnTF
2090       {
2091         \dim_abs:n
2092         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2093       }
2094       <
2095       { 0.001 pt }
2096       { \dim_use:N \l_@@_X_columns_dim }
2097       {
2098         \dim_eval:n
2099         {
2100           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2101           / \int_use:N \g_@@_total_X_weight_int
2102           + \l_@@_X_columns_dim
2103         }
2104       }
2105     }
2106   }
2107 }

```

11 Construction of the preamble of the array

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

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

```

2108 \cs_new_protected:Npn \@@_transform_preamble:
2109 {
2110   \@@_transform_preamble_i:
2111   \@@_transform_preamble_ii:
2112 }
2113 \cs_new_protected:Npn \@@_transform_preamble_i:
2114 {
2115   \int_gzero:N \c@jCol

```

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

```

2116   \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.

```

2117   \bool_gset_false:N \g_tmpb_bool

```

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

```
2118 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2119 \int_zero:N \l_tmpa_int
2120 \tl_gclear:N \g_@@_array_preamble_tl
2121 \str_if_eq:eeTF \l_@@_vlines_clist { all }
2122 {
2123   \tl_gset:Nn \g_@@_array_preamble_tl
2124     { ! { \skip_horizontal:N \arrayrulewidth } }
2125 }
2126 {
2127   \clist_if_in:NnT \l_@@_vlines_clist 1
2128   {
2129     \tl_gset:Nn \g_@@_array_preamble_tl
2130       { ! { \skip_horizontal:N \arrayrulewidth } }
2131   }
2132 }
```

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

```
2133 \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2134 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2135 \@@_replace_columncolor:
2136 }
```

```
2137 \hook_gput_code:nnn { begindocument } { . }
2138 {
2139   \IfPackageLoadedTF { colortbl }
2140   {
```

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).

```
2141   \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2142   \cs_new_protected:Npn \@@_replace_columncolor:
2143   {
2144     \regex_replace_all:NnN
2145       \c_@@_columncolor_regex
2146       { \c { @@_columncolor_preamble } }
2147       \g_@@_array_preamble_tl
2148   }
2149 }
2150 {
2151   \cs_new_protected:Npn \@@_replace_columncolor:
2152   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2153 }
2154 }
```

```
2155 \cs_new_protected:Npn \@@_transform_preamble_ii:
2156 {
```

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

```
2157 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2158 {
2159   \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2160   { \bool_gset_true:N \g_@@_delims_bool }
2161 }
2162 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2163 \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).

```
2164 \int_if_zero:nTF \l_@@_first_col_int
2165 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2166 {
2167   \bool_if:NF \g_@@_delims_bool
2168   {
2169     \bool_if:NF \l_@@_tabular_bool
2170     {
2171       \clist_if_empty:NT \l_@@_vlines_clist
2172       {
2173         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2174         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2175       }
2176     }
2177   }
2178 }
2179 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2180 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2181 {
2182   \bool_if:NF \g_@@_delims_bool
2183   {
2184     \bool_if:NF \l_@@_tabular_bool
2185     {
2186       \clist_if_empty:NT \l_@@_vlines_clist
2187       {
2188         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2189         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2190       }
2191     }
2192   }
2193 }
```

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).

```
2194 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2195 {
```

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

```
2196 \bool_if:NF \c_@@_testphase_table_bool
2197 {
2198   \tl_gput_right:Nn \g_@@_array_preamble_tl
2199   { > { \@@_error_too_much_cols: } l }
2200 }
2201 }
2202 }
```

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.

```
2203 \cs_new_protected:Npn \@@_rec_preamble:n #1
2204 {
```

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

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

```

2205 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2206 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2207 {

```

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

```

2208 \cs_if_exist:cTF { NC @ find @ #1 }
2209 {
2210 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2211 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2212 }
2213 {
2214 \str_if_eq:nnTF { #1 } { S }
2215 { \@@_fatal:n { unknown~column~type~S } }
2216 { \@@_fatal:nn { unknown~column~type } { #1 } }
2217 }
2218 }
2219 }

```

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

```

2220 \cs_new_protected:Npn \@@_c #1
2221 {
2222 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2223 \tl_gclear:N \g_@@_pre_cell_tl
2224 \tl_gput_right:Nn \g_@@_array_preamble_tl
2225 { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2226 \int_gincr:N \c@jCol
2227 \@@_rec_preamble_after_col:n
2228 }

2229 \cs_new_protected:Npn \@@_l #1
2230 {
2231 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2232 \tl_gclear:N \g_@@_pre_cell_tl
2233 \tl_gput_right:Nn \g_@@_array_preamble_tl
2234 {
2235 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2236 l
2237 < \@@_cell_end:
2238 }
2239 \int_gincr:N \c@jCol
2240 \@@_rec_preamble_after_col:n
2241 }

2242 \cs_new_protected:Npn \@@_r #1
2243 {
2244 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2245 \tl_gclear:N \g_@@_pre_cell_tl
2246 \tl_gput_right:Nn \g_@@_array_preamble_tl
2247 {
2248 > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2249 r
2250 < \@@_cell_end:
2251 }
2252 \int_gincr:N \c@jCol
2253 \@@_rec_preamble_after_col:n
2254 }

```

For `!` and `@`

```

2255 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2256 {
2257 \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2258 \@@_rec_preamble:n

```

```

2259 }
2260 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2261 \cs_new_protected:cpn { @@ _ | } #1
2262 {
\l_tmpa_int is the number of successive occurrences of |
2263   \int_incr:N \l_tmpa_int
2264   \@@_make_preamble_i_i:n
2265 }
2266 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2267 {
2268   \str_if_eq:nnTF { #1 } { | }
2269   { \use:c { @@ _ | } | }
2270   { \@@_make_preamble_i_ii:nn { } #1 }
2271 }
2272 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2273 {
2274   \str_if_eq:nnTF { #2 } { [ ] }
2275   { \@@_make_preamble_i_iii:nw { #1 } [ ] }
2276   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2277 }
2278 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2279 { \@@_make_preamble_i_iii:nn { #1 , #2 } }
2280 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2281 {
2282   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2283   \tl_gput_right:Ne \g_@@_array_preamble_tl
2284   {

```

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

```

2285   \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2286 }
2287 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2288 {
2289   \@@_vline:n
2290   {
2291     position = \int_eval:n { \c@jCol + 1 } ,
2292     multiplicity = \int_use:N \l_tmpa_int ,
2293     total-width = \dim_use:N \l_@@_rule_width_dim ,
2294     #2
2295   }

```

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.

```

2296   }
2297   \int_zero:N \l_tmpa_int
2298   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2299   \@@_rec_preamble:n #1
2300 }

2301 \cs_new_protected:cpn { @@ _ > } #1 #2
2302 {
2303   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2304   \@@_rec_preamble:n
2305 }

2306 \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.

```

2307 \keys_define:nn { nicematrix / p-column }
2308 {
2309   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2310   r .value_forbidden:n = true ,
2311   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2312   c .value_forbidden:n = true ,
2313   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2314   l .value_forbidden:n = true ,
2315   S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2316   S .value_forbidden:n = true ,
2317   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2318   p .value_forbidden:n = true ,
2319   t .meta:n = p ,
2320   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2321   m .value_forbidden:n = true ,
2322   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2323   b .value_forbidden:n = true
2324 }

```

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

```

2325 \cs_new_protected:Npn \@@_p #1
2326 {
2327   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2328   \@@_make_preamble_ii_i:n
2329 }
2330 \cs_set_eq:NN \@@_b \@@_p
2331 \cs_set_eq:NN \@@_m \@@_p
2332 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2333 {
2334   \str_if_eq:nnTF { #1 } { [ ]
2335     { \@@_make_preamble_ii_ii:w [ ]
2336       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2337     }
2338   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2339   { \@@_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.

```

2340 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2341 {

```

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).

```

2342   \str_set:Nn \l_@@_hpos_col_str { j }
2343   \@@_keys_p_column:n { #1 }
2344   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2345 }
2346 \cs_new_protected:Npn \@@_keys_p_column:n #1
2347 { \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.

```

2348 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2349 {
2350   \use:e
2351   {
2352     \@@_make_preamble_ii_v:nnnnnnnn
2353     { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }

```

```

2354         { \dim_eval:n { #1 } }
2355     {

```

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.

```

2356         \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2357         { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2358     {

```

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

```

2359         \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2360         { \str_lowercase:o \l_@@_hpos_col_str }
2361     }
2362     \IfPackageLoadedTF { ragged2e }
2363     {
2364         \str_case:on \l_@@_hpos_col_str
2365         {
2366             c { \exp_not:N \Centering }
2367             l { \exp_not:N \RaggedRight }
2368             r { \exp_not:N \RaggedLeft }
2369         }
2370     }
2371     {
2372         \str_case:on \l_@@_hpos_col_str
2373         {
2374             c { \exp_not:N \centering }
2375             l { \exp_not:N \raggedright }
2376             r { \exp_not:N \raggedleft }
2377         }
2378     }
2379     #3
2380 }
2381 { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2382 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2383 { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2384 { #2 }
2385 {
2386     \str_case:onF \l_@@_hpos_col_str
2387     {
2388         { j } { c }
2389         { si } { c }
2390     }

```

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

```

2391         { \str_lowercase:o \l_@@_hpos_col_str }
2392     }
2393 }

```

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

```

2394     \int_gincr:N \c@jCol
2395     \@@_rec_preamble_after_col:n
2396 }

```

#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*.

```

2397 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2398 {
2399   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2400   {
2401     \tl_gput_right:Nn \g_@@_array_preamble_tl
2402     { > \@@_test_if_empty_for_S: }
2403   }
2404   {
2405     \tl_gput_right:Nn \g_@@_array_preamble_tl
2406     { > \@@_test_if_empty: }
2407   }
2408   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2409   \tl_gclear:N \g_@@_pre_cell_tl
2410   \tl_gput_right:Nn \g_@@_array_preamble_tl
2411   {
2412     > {

```

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.

```

2413       \dim_set:Nn \l_@@_col_width_dim { #2 }
2414       \bool_if:NT \c_@@_testphase_table_bool
2415       { \tag_struct_begin:n { tag = Div } }
2416       \@@_cell_begin:

```

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

```

2417       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2418       \everypar
2419       {
2420         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2421         \everypar { }
2422       }
2423       \bool_if:NT \c_@@_testphase_table_bool \tagpdfpara0n

```

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

```

2424       #3

```

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

```

2425       \g_@@_row_style_tl
2426       \arraybackslash
2427       #5
2428     }
2429     #8
2430     < {
2431       #6

```

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

```

2432       \@finalstrut \@arstrutbox
2433       \use:c { end #7 }

```

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

```

2434       #4
2435       \@@_cell_end:
2436       \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2437     }
2438   }
2439 }

```

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

```
2440 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2441 {
```

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 `&`).

```
2442 \group_align_safe_begin:
2443 \peek_meaning:NTF &
2444 {
2445 \group_align_safe_end:
2446 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2447 }
```

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

```
2448 \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2449 \skip_horizontal:N \l_@@_col_width_dim
2450 }
2451 }
2452 { \group_align_safe_end: }
2453 }

2454 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2455 {
2456 \peek_meaning:NT \__siunitx_table_skip:n
2457 { \bool_gset_true:N \g_@@_empty_cell_bool }
2458 }
```

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

```
2459 \cs_new_protected:Npn \@@_center_cell_box:
2460 {
```

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

```
2461 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2462 {
2463 \int_compare:nNnT
2464 { \box_ht:N \l_@@_cell_box }
2465 >
```

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).

```
2466 { \box_ht:N \strutbox }
2467 {
2468 \hbox_set:Nn \l_@@_cell_box
2469 {
2470 \box_move_down:nn
2471 {
2472 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2473 + \baselineskip ) / 2
2474 }
2475 { \box_use:N \l_@@_cell_box }
2476 }
2477 }
2478 }
2479 }
```

For V (similar to the V of varwidth).

```

2480 \cs_new_protected:Npn \@@_V #1 #2
2481 {
2482   \str_if_eq:nnTF { #1 } { [ ] }
2483   { \@@_make_preamble_V_i:w [ ] }
2484   { \@@_make_preamble_V_i:w [ ] { #2 } }
2485 }
2486 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2487 { \@@_make_preamble_V_ii:nn { #1 } }
2488 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2489 {
2490   \str_set:Nn \l_@@_vpos_col_str { p }
2491   \str_set:Nn \l_@@_hpos_col_str { j }
2492   \@@_keys_p_column:n { #1 }
2493   \IfPackageLoadedTF { varwidth }
2494   { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2495   {
2496     \@@_error_or_warning:n { varwidth-not-loaded }
2497     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2498   }
2499 }

```

For w and W

```

2500 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2501 \cs_new_protected:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }

```

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

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

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

#4 is the width of the column.

```

2502 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2503 {
2504   \str_if_eq:nnTF { #3 } { s }
2505   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2506   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2507 }

```

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.

```

2508 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2509 {
2510   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2511   \tl_gclear:N \g_@@_pre_cell_tl
2512   \tl_gput_right:Nn \g_@@_array_preamble_tl
2513   {
2514     > {
2515       \dim_set:Nn \l_@@_col_width_dim { #2 }
2516       \@@_cell_begin:
2517       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2518     }
2519     c
2520     < {
2521       \@@_cell_end_for_w_s:
2522       #1
2523       \@@_adjust_size_box:
2524       \box_use_drop:N \l_@@_cell_box
2525     }
2526   }
2527   \int_gincr:N \c@jCol
2528   \@@_rec_preamble_after_col:n
2529 }

```

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

```

2530 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2531 {
2532   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2533   \tl_gclear:N \g_@@_pre_cell_tl
2534   \tl_gput_right:Nn \g_@@_array_preamble_tl
2535   {
2536     > {

```

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.

```

2537       \dim_set:Nn \l_@@_col_width_dim { #4 }
2538       \hbox_set:Nw \l_@@_cell_box
2539       \@@_cell_begin:
2540       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2541     }
2542     c
2543     < {
2544       \@@_cell_end:
2545       \hbox_set_end:
2546       #1
2547       \@@_adjust_size_box:
2548       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2549     }
2550   }

```

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

```

2551   \int_gincr:N \c@jCol
2552   \@@_rec_preamble_after_col:n
2553 }

```

```

2554 \cs_new_protected:Npn \@@_special_W:
2555 {
2556   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2557   { \@@_warning:n { W~warning } }
2558 }

```

For `S` (of `siunitx`).

```

2559 \cs_new_protected:Npn \@@_S #1 #2
2560 {
2561   \str_if_eq:nnTF { #2 } { [ ]
2562     { \@@_make_preamble_S:w [ ] }
2563     { \@@_make_preamble_S:w [ ] { #2 } }
2564   }
2565   \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2566   { \@@_make_preamble_S:i:n { #1 } }
2567   \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2568   {
2569     \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2570     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2571     \tl_gclear:N \g_@@_pre_cell_tl
2572     \tl_gput_right:Nn \g_@@_array_preamble_tl
2573     {
2574       > {

```

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

```

2575       \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
2576       \keys_set:nn { siunitx } { #1 }
2577       \@@_cell_begin:

```

```

2578         \siunitx_cell_begin:w
2579     }
2580     c
2581     <
2582     {
2583         \siunitx_cell_end:

```

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

```

2584         \tl_gput_right:Nx \g_@@_cell_after_hook_tl
2585         {
2586             \bool_if:NTF \l__siunitx_table_text_bool
2587                 \bool_set_true:N
2588                 \bool_set_false:N
2589             \l__siunitx_table_text_bool
2590         }
2591         \@@_cell_end:
2592     }
2593 }

```

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

```

2594     \int_gincr:N \c@jCol
2595     \@@_rec_preamble_after_col:n
2596 }

```

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

```

2597 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2598 {
2599     \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.

```

2600     \int_if_zero:nTF \c@jCol
2601     {
2602         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2603         {

```

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

```

2604             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2605             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2606             \@@_rec_preamble:n #2
2607         }
2608         {
2609             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2610             \@@_make_preamble_iv:nn { #1 } { #2 }
2611         }
2612     }
2613     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2614 }
2615 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2616 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2617 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2618 {
2619     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2620     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2621     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2622     {
2623         \@@_error:nn { delimiter~after~opening } { #2 }
2624         \@@_rec_preamble:n
2625     }
2626     { \@@_rec_preamble:n #2 }
2627 }

```

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

```
2628 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2629 { \use:c { @@ _ \token_to_str:N ( } }
```

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

```
2630 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2631 {
2632   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2633   \tl_if_in:nnTF { ) } \} { #2 }
2634   { \@@_make_preamble_v:nnn #1 #2 }
2635   {
2636     \str_if_eq:nnTF { \@@_stop: } { #2 }
2637     {
2638       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2639       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2640       {
2641         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2642         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2643         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2644         \@@_rec_preamble:n #2
2645       }
2646     }
2647     {
2648       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2649       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2650       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2651       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2652       \@@_rec_preamble:n #2
2653     }
2654   }
2655 }
2656 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2657 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2658 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2659 {
2660   \str_if_eq:nnTF { \@@_stop: } { #3 }
2661   {
2662     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2663     {
2664       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2665       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2666       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2667       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2668     }
2669     {
2670       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2671       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2672       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2673       \@@_error:nn { double~closing~delimiter } { #2 }
2674     }
2675   }
2676   {
2677     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2678     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2679     \@@_error:nn { double~closing~delimiter } { #2 }
2680     \@@_rec_preamble:n #3
2681   }
2682 }
```

```

2683 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2684 { \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 @{...}.

```

2685 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2686 {
2687   \str_if_eq:nnTF { #1 } { < }
2688   \@@_rec_preamble_after_col_i:n
2689   {
2690     \str_if_eq:nnTF { #1 } { @ }
2691     \@@_rec_preamble_after_col_ii:n
2692     {
2693       \str_if_eq:eeTF \l_@@_vlines_clist { all }
2694       {
2695         \tl_gput_right:Nn \g_@@_array_preamble_tl
2696         { ! { \skip_horizontal:N \arrayrulewidth } }
2697       }
2698       {
2699         \clist_if_in:NeT \l_@@_vlines_clist
2700         { \int_eval:n { \c@jCol + 1 } }
2701         {
2702           \tl_gput_right:Nn \g_@@_array_preamble_tl
2703           { ! { \skip_horizontal:N \arrayrulewidth } }
2704         }
2705       }
2706       \@@_rec_preamble:n { #1 }
2707     }
2708   }
2709 }
2710 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2711 {
2712   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2713   \@@_rec_preamble_after_col:n
2714 }

```

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.

```

2715 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2716 {
2717   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2718   {
2719     \tl_gput_right:Nn \g_@@_array_preamble_tl
2720     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2721   }
2722   {
2723     \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2724     {
2725       \tl_gput_right:Nn \g_@@_array_preamble_tl
2726       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2727     }
2728     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2729   }
2730   \@@_rec_preamble:n
2731 }
2732 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2733 {
2734   \tl_clear:N \l_tmpa_tl
2735   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2736   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2737 }

```

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.

```
2738 \cs_new_protected:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }
```

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

```
2739 \cs_new_protected:Npn \@@_X #1 #2
2740 {
2741   \str_if_eq:nnTF { #2 } { [ ]
2742     { \@@_make_preamble_X:w [ ] }
2743     { \@@_make_preamble_X:w [ ] #2 }
2744   }
2745   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2746   { \@@_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`).

```
2747 \keys_define:nn { nicematrix / X-column }
2748 { 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.

```
2749 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2750 {
```

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).

```
2751   \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).

```
2752   \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`.

```
2753   \int_zero_new:N \l_@@_weight_int
2754   \int_set_eq:NN \l_@@_weight_int \c_one_int
2755   \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2756   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2757   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2758   {
2759     \@@_error_or_warning:n { negative-weight }
2760     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2761   }
2762   \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).

```
2763   \bool_if:NNTF \l_@@_X_columns_aux_bool
2764   {
2765     \@@_make_preamble_ii_iv:nnn
2766     { \l_@@_weight_int \l_@@_X_columns_dim }
2767     { minipage }
2768     { \@@_no_update_width: }
2769   }
2770   {
2771     \tl_gput_right:Nn \g_@@_array_preamble_tl
2772     {
```



```

2773 > {
2774     \@@_cell_begin:
2775     \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`.

```

2776     \NotEmpty

```

The following code will nullify the box of the cell.

```

2777     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2778     { \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`.

```

2779     \begin { minipage } { 5 cm } \arraybackslash
2780     }
2781     c
2782     < {
2783         \end { minipage }
2784         \@@_cell_end:
2785     }
2786     }
2787     \int_gincr:N \c@jCol
2788     \@@_rec_preamble_after_col:n
2789 }
2790 }

```

```

2791 \cs_new_protected:Npn \@@_no_update_width:
2792 {
2793     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2794     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2795 }

```

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

```

2796 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2797 {
2798     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2799     { \int_eval:n { \c@jCol + 1 } }
2800     \tl_gput_right:Ne \g_@@_array_preamble_tl
2801     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2802     \@@_rec_preamble:n
2803 }

```

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.

```

2804 \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).

```

2805 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2806 { \@@_fatal:n { Preamble-forgotten } }
2807 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2808 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2809 \cs_set_eq:cc { @@ _ \token_to_str:N \Block } { @@ _ \token_to_str:N \hline }
2810 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
2811 \cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle } { @@ _ \token_to_str:N \hline }
2812 \cs_set_eq:cc { @@ _ \token_to_str:N \diagbox } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```
2813 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2814 {
```

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`.

```
2815 \multispan { #1 }
2816 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2817 \begingroup
2818 \bool_if:NT \c_@@_testphase_table_bool
2819 { \tbl_update_multicolumn_cell_data:n { #1 } }
2820 \cs_set_nopar:Npn \@addamp
2821 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2822 \tl_gclear:N \g_@@_preamble_tl
2823 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2824 \exp_args:No \mkpream \g_@@_preamble_tl
2825 \@addtopreamble \empty
2826 \endgroup
2827 \bool_if:NT \c_@@_recent_array_bool
2828 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2829 \int_compare:nNnT { #1 } > \c_one_int
2830 {
2831 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2832 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2833 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2834 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2835 {
2836 {
2837 \int_if_zero:nTF \c@jCol
2838 { \int_eval:n { \c@iRow + 1 } }
2839 { \int_use:N \c@iRow }
2840 }
2841 { \int_eval:n { \c@jCol + 1 } }
2842 {
2843 \int_if_zero:nTF \c@jCol
2844 { \int_eval:n { \c@iRow + 1 } }
2845 { \int_use:N \c@iRow }
2846 }
2847 { \int_eval:n { \c@jCol + #1 } }
2848 { } % for the name of the block
2849 }
2850 }
```

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

```
2851 \RenewDocumentCommand \cellcolor { 0 { } m }
2852 {
2853 \tl_gput_right:Ne \g_@@_pre_code_before_tl
2854 {
2855 \@@_rectanglecolor [ ##1 ]
2856 { \exp_not:n { ##2 } }
```

```

2857         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2858         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2859     }
2860     \ignorespaces
2861 }

```

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

```

2862     \cs_set_nopar:Npn \@sharp { #3 }
2863     \@arstrut
2864     \@preamble
2865     \null

```

We add some lines.

```

2866     \int_gadd:Nn \c@jCol { #1 - 1 }
2867     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2868         { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2869     \ignorespaces
2870 }

```

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`.

```

2871 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2872 {
2873     \str_case:nnF { #1 }
2874     {
2875         c { \@@_make_m_preamble_i:n #1 }
2876         l { \@@_make_m_preamble_i:n #1 }
2877         r { \@@_make_m_preamble_i:n #1 }
2878         > { \@@_make_m_preamble_ii:nn #1 }
2879         ! { \@@_make_m_preamble_ii:nn #1 }
2880         @ { \@@_make_m_preamble_ii:nn #1 }
2881         | { \@@_make_m_preamble_iii:n #1 }
2882         p { \@@_make_m_preamble_iv:nnn t #1 }
2883         m { \@@_make_m_preamble_iv:nnn c #1 }
2884         b { \@@_make_m_preamble_iv:nnn b #1 }
2885         w { \@@_make_m_preamble_v:nnnn { } #1 }
2886         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2887         \q_stop { }
2888     }
2889     {
2890         \cs_if_exist:cTF { NC @ find @ #1 }
2891         {
2892             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2893             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2894         }
2895         {
2896             \str_if_eq:nnTF { #1 } { S }
2897             { \@@_fatal:n { unknown~column~type~S } }
2898             { \@@_fatal:nn { unknown~column~type } { #1 } }
2899         }
2900     }
2901 }

```

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

```

2902 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2903 {
2904     \tl_gput_right:Nn \g_@@_preamble_tl
2905     {
2906         > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2907         #1
2908         < \@@_cell_end:
2909     }

```

We test for the presence of a <.

```
2910 \@@_make_m_preamble_x:n
2911 }
```

For >, ! and @

```
2912 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2913 {
2914 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2915 \@@_make_m_preamble:n
2916 }
```

For |

```
2917 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2918 {
2919 \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2920 \@@_make_m_preamble:n
2921 }
```

For p, m and b

```
2922 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2923 {
2924 \tl_gput_right:Nn \g_@@_preamble_tl
2925 {
2926 > {
2927 \@@_cell_begin:
2928 \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2929 \mode_leave_vertical:
2930 \arraybackslash
2931 \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2932 }
2933 c
2934 < {
2935 \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2936 \end { minipage }
2937 \@@_cell_end:
2938 }
2939 }
```

We test for the presence of a <.

```
2940 \@@_make_m_preamble_x:n
2941 }
```

For w and W

```
2942 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2943 {
2944 \tl_gput_right:Nn \g_@@_preamble_tl
2945 {
2946 > {
2947 \dim_set:Nn \l_@@_col_width_dim { #4 }
2948 \hbox_set:Nw \l_@@_cell_box
2949 \@@_cell_begin:
2950 \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2951 }
2952 c
2953 < {
2954 \@@_cell_end:
2955 \hbox_set_end:
2956 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2957 #1
2958 \@@_adjust_size_box:
2959 \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2960 }
2961 }
```

We test for the presence of a <.

```
2962 \@@_make_m_preamble_x:n
2963 }
```

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

```
2964 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2965 {
2966   \str_if_eq:nnTF { #1 } { < }
2967     \@@_make_m_preamble_ix:n
2968     { \@@_make_m_preamble:n { #1 } }
2969 }
2970 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2971 {
2972   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2973   \@@_make_m_preamble_x:n
2974 }
```

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`).

```
2975 \cs_new_protected:Npn \@@_put_box_in_flow:
2976 {
2977   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2978   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2979   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2980     { \box_use_drop:N \l_tmpa_box }
2981   \@@_put_box_in_flow_i:
2982 }
```

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

```
2983 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2984 {
2985   \pgfpicture
2986     \@@_qpoint:n { row - 1 }
2987     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2988     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2989     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2990     \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).

```
2991 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2992 {
2993   \int_set:Nn \l_tmpa_int
2994     {
2995       \str_range:Nnn
2996         \l_@@_baseline_tl
2997         6
2998         { \tl_count:o \l_@@_baseline_tl }
2999     }
3000   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3001 }
3002 {
3003   \str_if_eq:eeTF \l_@@_baseline_tl { t }
3004     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3005     {
3006       \str_if_eq:onTF \l_@@_baseline_tl { b }
3007         { \int_set_eq:NN \l_tmpa_int \c@iRow }
3008         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3009     }
```

```

3009     }
3010     \bool_lazy_or:nnT
3011     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3012     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3013     {
3014         \@@_error:n { bad-value-for-baseline }
3015         \int_set_eq:NN \l_tmpa_int \c_one_int
3016     }
3017     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3018     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3019     }
3020     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3021     \endpgfpicture
3022     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3023     \box_use_drop:N \l_tmpa_box
3024 }

```

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).

```

3025 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3026 {

```

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.

```

3027     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3028     {
3029         \int_compare:nNnT \c_jCol > \c_one_int
3030         {
3031             \box_set_wd:Nn \l_@@_the_array_box
3032             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3033         }
3034     }

```

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

```

3035     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3036     \bool_if:NT \l_@@_caption_above_bool
3037     {
3038         \tl_if_empty:NF \l_@@_caption_tl
3039         {
3040             \bool_set_false:N \g_@@_caption_finished_bool
3041             \int_gzero:N \c@tabularnote
3042             \@@_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).

```

3043         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3044         {
3045             \tl_gput_right:Ne \g_@@_aux_tl
3046             {
3047                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3048                 { \int_use:N \g_@@_notes_caption_int }
3049             }
3050             \int_gzero:N \g_@@_notes_caption_int
3051         }
3052     }
3053 }

```

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

```

3054   \hbox
3055   {
3056   \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.

```

3057   \@@_create_extra_nodes:
3058   \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3059 }

```

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

```

3060   \bool_lazy_any:nT
3061   {
3062     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3063     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3064     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3065   }
3066   \@@_insert_tabularnotes:
3067   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3068   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3069   \end { minipage }
3070 }

```

```

3071 \cs_new_protected:Npn \@@_insert_caption:
3072 {
3073   \tl_if_empty:NF \l_@@_caption_tl
3074   {
3075     \cs_if_exist:NTF \@caption
3076     { \@@_insert_caption_i: }
3077     { \@@_error:n { caption~outside~float } }
3078   }
3079 }

```

```

3080 \cs_new_protected:Npn \@@_insert_caption_i:
3081 {
3082   \group_begin:

```

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

```

3083   \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.

```

3084   \IfPackageLoadedT { floatrow }
3085   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3086   \tl_if_empty:NTF \l_@@_short_caption_tl
3087   { \caption }
3088   { \caption [ \l_@@_short_caption_tl ] }
3089   { \l_@@_caption_tl }

```

In some circonstancies (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.

```

3090 \bool_if:NF \g_@@_caption_finished_bool
3091 {
3092     \bool_gset_true:N \g_@@_caption_finished_bool
3093     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3094     \int_gzero:N \c@tabularnote
3095 }
3096 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3097 \group_end:
3098 }
3099 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3100 {
3101     \@@_error_or_warning:n { tabularnote~below~the~tabular }
3102     \@@_gredirect_none:n { tabularnote~below~the~tabular }
3103 }
3104 \cs_new_protected:Npn \@@_insert_tabularnotes:
3105 {
3106     \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3107     \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3108     \skip_vertical:N 0.65ex

```

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

```

3109 \group_begin:
3110 \l_@@_notes_code_before_tl
3111 \tl_if_empty:NF \g_@@_tabularnote_tl
3112 {
3113     \g_@@_tabularnote_tl \par
3114     \tl_gclear:N \g_@@_tabularnote_tl
3115 }

```

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.

```

3116 \int_compare:nNnT \c@tabularnote > \c_zero_int
3117 {
3118     \bool_if:NTF \l_@@_notes_para_bool
3119     {
3120         \begin { tabularnotes* }
3121         \seq_map_inline:Nn \g_@@_notes_seq
3122             { \@@_one_tabularnote:nn ##1 }
3123         \strut
3124         \end { tabularnotes* }

```

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

```

3125 \par
3126 }
3127 {
3128     \tabularnotes
3129     \seq_map_inline:Nn \g_@@_notes_seq
3130         { \@@_one_tabularnote:nn ##1 }
3131     \strut
3132     \endtabularnotes
3133 }
3134 }
3135 \unskip
3136 \group_end:
3137 \bool_if:NT \l_@@_notes_bottomrule_bool
3138 {
3139     \IfPackageLoadedTF { booktabs }
3140     {

```

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

```

3141 \skip_vertical:N \aboverulesep

```


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

```

3142         { \CT@arc@ \hrule height \heavyrulewidth }
3143     }
3144     { \@_error_or_warning:n { bottomrule-without-booktabs } }
3145 }
3146 \l_@@_notes_code_after_tl
3147 \seq_gclear:N \g_@@_notes_seq
3148 \seq_gclear:N \g_@@_notes_in_caption_seq
3149 \int_gzero:N \c@tabularnote
3150 }

```

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

```

3151 \cs_set_protected:Npn \@_one_tabularnote:nn #1
3152 {
3153     \tl_if_novalue:nTF { #1 }
3154     { \item }
3155     { \item [ \@_notes_label_in_list:n { #1 } ] }
3156 }

```

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.

```

3157 \cs_new_protected:Npn \@_use_arraybox_with_notes_b:
3158 {
3159     \pgfpicture
3160     \@_qpoint:n { row - 1 }
3161     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3162     \@_qpoint:n { row - \int_use:N \c@iRow - base }
3163     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3164     \endpgfpicture
3165     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3166     \int_if_zero:nT \l_@@_first_row_int
3167     {
3168         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3169         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3170     }
3171     \box_move_up:nn \g_tmpa_dim { \hbox { \@_use_arraybox_with_notes_c: } }
3172 }

```

Now, the general case.

```

3173 \cs_new_protected:Npn \@_use_arraybox_with_notes:
3174 {

```

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

```

3175     \str_if_eq:eeT \l_@@_baseline_tl { t }
3176     { \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`.

```

3177     \pgfpicture
3178     \@_qpoint:n { row - 1 }
3179     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3180     \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3181     {
3182         \int_set:Nn \l_tmpa_int
3183         {
3184             \str_range:Nnn
3185             \l_@@_baseline_tl
3186             6
3187             { \tl_count:o \l_@@_baseline_tl }

```

```

3188     }
3189     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3190 }
3191 {
3192     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3193     \bool_lazy_or:nnT
3194     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3195     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3196     {
3197         \@@_error:n { bad-value-for-baseline }
3198         \int_set:Nn \l_tmpa_int 1
3199     }
3200     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3201 }
3202 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3203 \endpgfpicture
3204 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3205 \int_if_zero:nT \l_@@_first_row_int
3206 {
3207     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3208     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3209 }
3210 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3211 }

```

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.

```

3212 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3213 {

```

We will compute the real width of both delimiters used.

```

3214     \dim_zero_new:N \l_@@_real_left_delim_dim
3215     \dim_zero_new:N \l_@@_real_right_delim_dim
3216     \hbox_set:Nn \l_tmpb_box
3217     {
3218         \m@th % added 2024/11/21
3219         \c_math_toggle_token
3220         \left #1
3221         \vcenter
3222         {
3223             \vbox_to_ht:nn
3224             { \box_ht_plus_dp:N \l_tmpa_box }
3225             { }
3226         }
3227         \right .
3228         \c_math_toggle_token
3229     }
3230     \dim_set:Nn \l_@@_real_left_delim_dim
3231     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3232     \hbox_set:Nn \l_tmpb_box
3233     {
3234         \m@th % added 2024/11/21
3235         \c_math_toggle_token
3236         \left .
3237         \vbox_to_ht:nn
3238         { \box_ht_plus_dp:N \l_tmpa_box }
3239         { }
3240         \right #2
3241         \c_math_toggle_token
3242     }
3243     \dim_set:Nn \l_@@_real_right_delim_dim
3244     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3245 \skip_horizontal:N \l_@@_left_delim_dim
3246 \skip_horizontal:N -\l_@@_real_left_delim_dim
3247 \@@_put_box_in_flow:
3248 \skip_horizontal:N \l_@@_right_delim_dim
3249 \skip_horizontal:N -\l_@@_real_right_delim_dim
3250 }

```

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).

```

3251 \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).

```

3252 {
3253   \peek_remove_spaces:n
3254   {
3255     \peek_meaning:NTF \end
3256     \@@_analyze_end:Nn
3257     {
3258       \@@_transform_preamble:

```

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

```

3259       \@@_array:o \g_@@_array_preamble_tl
3260     }
3261   }
3262 }
3263 {
3264   \@@_create_col_nodes:
3265   \endarray
3266 }

```

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

```

3267 \NewDocumentEnvironment { @@-light-syntax } { b }
3268 {

```

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`.

```

3269 \tl_if_empty:nT { #1 }
3270 { \@@_fatal:n { empty-environment } }
3271 \tl_if_in:nnT { #1 } { & }
3272 { \@@_fatal:n { ampersand-in~light-syntax } }
3273 \tl_if_in:nnT { #1 } { \ }
3274 { \@@_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`.

```

3275 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3276 }

```

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.

```

3277 {
3278   \@@_create_col_nodes:
3279   \endarray
3280 }

3281 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3282 {
3283   \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).

```

3284   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3285   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3286   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3287     \seq_set_split:Nee
3288     \seq_set_split:Non
3289     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3290   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3291   \tl_if_empty:NF \l_tmpa_tl
3292   { \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.

```

3293   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3294     { \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`).

```

3295   \tl_build_begin:N \l_@@_new_body_tl
3296   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3297   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3298   \@@_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).

```

3299   \seq_map_inline:Nn \l_@@_rows_seq
3300   {
3301     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3302     \@@_line_with_light_syntax:n { ##1 }
3303   }
3304   \tl_build_end:N \l_@@_new_body_tl

3305   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3306   {
3307     \int_set:Nn \l_@@_last_col_int
3308     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3309   }

```

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.

```

3310   \@@_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`).

```

3311   \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3312 }

```

```

3313 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3314 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3315 {
3316   \seq_clear_new:N \l_@@_cells_seq
3317   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3318   \int_set:Nn \l_@@_nb_cols_int
3319   {
3320     \int_max:nn
3321     \l_@@_nb_cols_int
3322     { \seq_count:N \l_@@_cells_seq }
3323   }
3324   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3325   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3326   \seq_map_inline:Nn \l_@@_cells_seq
3327   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3328 }

```

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`.

```

3329 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3330 {
3331   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3332   { \@@_fatal:n { empty-environment } }

```

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

```

3333   \end { #2 }
3334 }

```

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`).

```

3335 \cs_new:Npn \@@_create_col_nodes:
3336 {
3337   \crrr
3338   \int_if_zero:nT \l_@@_first_col_int
3339   {
3340     \omit
3341     \hbox_overlap_left:n
3342     {
3343       \bool_if:NT \l_@@_code_before_bool
3344       { \pgfsys@markposition { \@@_env: - col - 0 } }
3345       \pgfpicture
3346       \pgfrememberpicturepositiononpagetrue
3347       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3348       \str_if_empty:NF \l_@@_name_str
3349       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3350       \endpgfpicture
3351       \skip_horizontal:N 2\col@sep
3352       \skip_horizontal:N \g_@@_width_first_col_dim
3353     }
3354     &
3355   }
3356   \omit

```

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

```

3357   \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`).

```

3358   \int_if_zero:nTF \l_@@_first_col_int
3359   {

```

```

3360 \bool_if:NT \l_@@_code_before_bool
3361 {
3362   \hbox
3363   {
3364     \skip_horizontal:N -0.5\arrayrulewidth
3365     \pgfsys@markposition { \@@_env: - col - 1 }
3366     \skip_horizontal:N 0.5\arrayrulewidth
3367   }
3368 }
3369 \pgfpicture
3370 \pgfrememberpicturepositiononpagetrue
3371 \pgfcoordinate { \@@_env: - col - 1 }
3372 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3373 \str_if_empty:NF \l_@@_name_str
3374 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3375 \endpgfpicture
3376 }
3377 {
3378 \bool_if:NT \l_@@_code_before_bool
3379 {
3380   \hbox
3381   {
3382     \skip_horizontal:N 0.5\arrayrulewidth
3383     \pgfsys@markposition { \@@_env: - col - 1 }
3384     \skip_horizontal:N -0.5\arrayrulewidth
3385   }
3386 }
3387 \pgfpicture
3388 \pgfrememberpicturepositiononpagetrue
3389 \pgfcoordinate { \@@_env: - col - 1 }
3390 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3391 \str_if_empty:NF \l_@@_name_str
3392 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3393 \endpgfpicture
3394 }

```

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.

```

3395 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3396 \bool_if:NF \l_@@_auto_columns_width_bool
3397 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3398 {
3399   \bool_lazy_and:nnTF
3400   \l_@@_auto_columns_width_bool
3401   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3402   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3403   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3404   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3405 }
3406 \skip_horizontal:N \g_tmpa_skip
3407 \hbox
3408 {
3409   \bool_if:NT \l_@@_code_before_bool
3410   {
3411     \hbox
3412     {
3413       \skip_horizontal:N -0.5\arrayrulewidth
3414       \pgfsys@markposition { \@@_env: - col - 2 }
3415       \skip_horizontal:N 0.5\arrayrulewidth

```

```

3416     }
3417   }
3418   \pgfpicture
3419   \pgfrememberpicturepositiononpagetrue
3420   \pgfcoordinate { \@@_env: - col - 2 }
3421   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3422   \str_if_empty:NF \l_@@_name_str
3423   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3424   \endpgfpicture
3425 }

```

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.

```

3426   \int_gset_eq:NN \g_tmpa_int \c_one_int
3427   \bool_if:NTF \g_@@_last_col_found_bool
3428   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3429   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3430   {
3431     &
3432     \omit
3433     \int_gincr:N \g_tmpa_int

```

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

```

3434     \skip_horizontal:N \g_tmpa_skip
3435     \bool_if:NT \l_@@_code_before_bool
3436     {
3437       \hbox
3438       {
3439         \skip_horizontal:N -0.5\arrayrulewidth
3440         \pgfsys@markposition
3441         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3442         \skip_horizontal:N 0.5\arrayrulewidth
3443       }
3444     }

```

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

```

3445   \pgfpicture
3446   \pgfrememberpicturepositiononpagetrue
3447   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3448   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3449   \str_if_empty:NF \l_@@_name_str
3450   {
3451     \pgfnodealias
3452     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3453     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3454   }
3455   \endpgfpicture
3456 }

3457 &
3458 \omit

```

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

```

3459   \int_if_zero:nT \g_@@_col_total_int
3460   { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3461   \skip_horizontal:N \g_tmpa_skip
3462   \int_gincr:N \g_tmpa_int
3463   \bool_lazy_any:nF
3464   {
3465     \g_@@_delims_bool
3466     \l_@@_tabular_bool
3467     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3468     \l_@@_exterior_arraycolsep_bool

```

```

3469         \l_@@_bar_at_end_of_pream_bool
3470     }
3471     { \skip_horizontal:N -\col@sep }
3472 \bool_if:NT \l_@@_code_before_bool
3473 {
3474     \hbox
3475     {
3476         \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.

```

3477         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3478     { \skip_horizontal:N -\arraycolsep }
3479     \pgfsys@markposition
3480     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3481     \skip_horizontal:N 0.5\arrayrulewidth
3482     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3483     { \skip_horizontal:N \arraycolsep }
3484 }
3485 }
3486 \pgfpicture
3487 \pgfrememberpicturerepositiononpagetrue
3488 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489 {
3490     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3491     {
3492         \pgfpoint
3493         { - 0.5 \arrayrulewidth - \arraycolsep }
3494         \c_zero_dim
3495     }
3496     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3497 }
3498 \str_if_empty:NF \l_@@_name_str
3499 {
3500     \pgfnodealias
3501     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3502     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3503 }
3504 \endpgfpicture

3505 \bool_if:NT \g_@@_last_col_found_bool
3506 {
3507     \hbox_overlap_right:n
3508     {
3509         \skip_horizontal:N \g_@@_width_last_col_dim
3510         \skip_horizontal:N \col@sep
3511         \bool_if:NT \l_@@_code_before_bool
3512         {
3513             \pgfsys@markposition
3514             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3515         }
3516         \pgfpicture
3517         \pgfrememberpicturerepositiononpagetrue
3518         \pgfcoordinate
3519         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3520         \pgfpintorigin
3521         \str_if_empty:NF \l_@@_name_str
3522         {
3523             \pgfnodealias
3524             {
3525                 \l_@@_name_str - col
3526                 - \int_eval:n { \g_@@_col_total_int + 1 }

```



```

3527         }
3528         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3529     }
3530     \endpgfpicture
3531 }
3532 }
3533 % \cr
3534 }

```

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

```

3535 \tl_const:Nn \c_@@_preamble_first_col_tl
3536 {
3537     >
3538     {

```

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

```

3539         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3540         \bool_gset_true:N \g_@@_after_col_zero_bool
3541         \@@_begin_of_row:
3542         \hbox_set:Nw \l_@@_cell_box
3543         \@@_math_toggle:
3544         \@@_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”.

```

3545         \int_compare:nNnT \c@iRow > \c_zero_int
3546         {
3547             \bool_lazy_or:nnT
3548             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3549             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3550             {
3551                 \l_@@_code_for_first_col_tl
3552                 \xglobal \colorlet { nicematrix-first-col } { . }
3553             }
3554         }
3555     }

```

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`.

```

3556     l
3557     <
3558     {
3559         \@@_math_toggle:
3560         \hbox_set_end:
3561         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3562         \@@_adjust_size_box:
3563         \@@_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.

```

3564         \dim_gset:Nn \g_@@_width_first_col_dim
3565         { \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.

```

3566         \hbox_overlap_left:n
3567         {
3568             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3569             \@@_node_for_cell:
3570             { \box_use_drop:N \l_@@_cell_box }
3571             \skip_horizontal:N \l_@@_left_delim_dim
3572             \skip_horizontal:N \l_@@_left_margin_dim
3573             \skip_horizontal:N \l_@@_extra_left_margin_dim
3574         }

```

```

3575     \bool_gset_false:N \g_@@_empty_cell_bool
3576     \skip_horizontal:N -2\col@sep
3577   }
3578 }

```

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

```

3579 \tl_const:Nn \c_@@_preamble_last_col_tl
3580 {
3581   >
3582   {
3583     \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).

```

3584     \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.

```

3585     \bool_gset_true:N \g_@@_last_col_found_bool
3586     \int_gincr:N \c@jCol
3587     \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3588     \hbox_set:Nw \l_@@_cell_box
3589     \@@_math_toggle:
3590     \@@_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”.

```

3591     \int_compare:nNnT \c@iRow > \c_zero_int
3592     {
3593       \bool_lazy_or:nnT
3594       { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3595       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3596       {
3597         \l_@@_code_for_last_col_tl
3598         \xglobal \colorlet { nicematrix-last-col } { . }
3599       }
3600     }
3601   }
3602   l
3603   <
3604   {
3605     \@@_math_toggle:
3606     \hbox_set_end:
3607     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3608     \@@_adjust_size_box:
3609     \@@_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.

```

3610     \dim_gset:Nn \g_@@_width_last_col_dim
3611     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3612     \skip_horizontal:N -2\col@sep

```

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

```

3613     \hbox_overlap_right:n
3614     {
3615       \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3616       {
3617         \skip_horizontal:N \l_@@_right_delim_dim
3618         \skip_horizontal:N \l_@@_right_margin_dim
3619         \skip_horizontal:N \l_@@_extra_right_margin_dim
3620         \@@_node_for_cell:
3621       }
3622     }
3623     \bool_gset_false:N \g_@@_empty_cell_bool
3624   }
3625 }

```

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

```

3626 \NewDocumentEnvironment { NiceArray } { }
3627 {
3628   \bool_gset_false:N \g_@@_delims_bool
3629   \str_if_empty:NT \g_@@_name_env_str
3630   { \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).

```

3631   \NiceArrayWithDelims . .
3632 }
3633 { \endNiceArrayWithDelims }

```

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

```

3634 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3635 {
3636   \NewDocumentEnvironment { #1 NiceArray } { }
3637   {
3638     \bool_gset_true:N \g_@@_delims_bool
3639     \str_if_empty:NT \g_@@_name_env_str
3640     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3641     \@@_test_if_math_mode:
3642     \NiceArrayWithDelims #2 #3
3643   }
3644   { \endNiceArrayWithDelims }
3645 }
3646 \@@_def_env:nnn p ( )
3647 \@@_def_env:nnn b [ ]
3648 \@@_def_env:nnn B \{ \}
3649 \@@_def_env:nnn v | |
3650 \@@_def_env:nnn V \l \r

```

13 The environment `{NiceMatrix}` and its variants

```

3651 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3652 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3653 {
3654   \bool_set_false:N \l_@@_preamble_bool
3655   \tl_clear:N \l_tmpa_tl
3656   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3657   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3658   \tl_put_right:Nn \l_tmpa_tl
3659   {
3660     *
3661     {
3662       \int_case:nnF \l_@@_last_col_int
3663       {
3664         -2 { \c@MaxMatrixCols }
3665         -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).

```

3666       }
3667       { \int_eval:n { \l_@@_last_col_int - 1 } }
3668     }
3669     { #2 }
3670   }
3671   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3672   \exp_args:No \l_tmpb_tl \l_tmpa_tl

```

```

3673 }
3674 \clist_map_inline:nn { p , b , B , v , V }
3675 {
3676   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3677   {
3678     \bool_gset_true:N \g_@@_delims_bool
3679     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3680     \int_if_zero:nT \l_@@_last_col_int
3681     {
3682       \bool_set_true:N \l_@@_last_col_without_value_bool
3683       \int_set:Nn \l_@@_last_col_int { -1 }
3684     }
3685     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3686     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3687   }
3688   { \use:c { end #1 NiceArray } }
3689 }

```

We define also an environment {NiceMatrix}

```

3690 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3691 {
3692   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3693   \int_if_zero:nT \l_@@_last_col_int
3694   {
3695     \bool_set_true:N \l_@@_last_col_without_value_bool
3696     \int_set:Nn \l_@@_last_col_int { -1 }
3697   }
3698   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3699   \bool_lazy_or:nnT
3700   { \clist_if_empty_p:N \l_@@_vlines_clist }
3701   { \l_@@_except_borders_bool }
3702   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3703   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3704 }
3705 { \endNiceArray }

```

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

```

3706 \cs_new_protected:Npn \@@_NotEmpty:
3707 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3708 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3709 {

```

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

```

3710   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3711   { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3712   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3713   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3714   \tl_if_empty:NF \l_@@_short_caption_tl
3715   {
3716     \tl_if_empty:NT \l_@@_caption_tl
3717     {
3718       \@@_error_or_warning:n { short-caption-without~caption }
3719       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3720     }
3721   }
3722   \tl_if_empty:NF \l_@@_label_tl
3723   {

```

```

3724     \tl_if_empty:NT \l_@@_caption_tl
3725     { \@@_error_or_warning:n { label~without~caption } }
3726   }
3727   \NewDocumentEnvironment { TabularNote } { b }
3728   {
3729     \bool_if:NTF \l_@@_in_code_after_bool
3730     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3731     {
3732       \tl_if_empty:NF \g_@@_tabularnote_tl
3733       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3734       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3735     }
3736   }
3737   { }
3738   \@@_settings_for_tabular:
3739   \NiceArray { #2 }
3740 }
3741 { \endNiceArray }
3742 \cs_new_protected:Npn \@@_settings_for_tabular:
3743 {
3744   \bool_set_true:N \l_@@_tabular_bool
3745   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3746   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3747   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3748 }

3749 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3750 {
3751   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3752   \dim_zero_new:N \l_@@_width_dim
3753   \dim_set:Nn \l_@@_width_dim { #1 }
3754   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3755   \@@_settings_for_tabular:
3756   \NiceArray { #3 }
3757 }
3758 {
3759   \endNiceArray
3760   \int_if_zero:nT \g_@@_total_X_weight_int
3761   { \@@_error:n { NiceTabularX~without~X } }
3762 }

3763 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3764 {
3765   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3766   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3767   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3768   \@@_settings_for_tabular:
3769   \NiceArray { #3 }
3770 }
3771 { \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`).

```

3772 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3773 {
3774   \bool_lazy_all:nT

```

```

3775 {
3776   { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3777   \l_@@_hvlines_bool
3778   { ! \g_@@_delims_bool }
3779   { ! \l_@@_except_borders_bool }
3780 }
3781 {
3782   \bool_set_true:N \l_@@_except_borders_bool
3783   \clist_if_empty:NF \l_@@_corners_clist
3784   { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3785   \tl_gput_right:Nn \g_@@_pre_code_after_tl
3786   {
3787     \@@_stroke_block:nnn
3788     {
3789       rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3790       draw = \l_@@_rules_color_tl
3791     }
3792     { 1-1 }
3793     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3794   }
3795 }
3796 }

3797 \cs_new_protected:Npn \@@_after_array:
3798 {

```

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`.

```

3799   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3800   \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.

```

3801   \bool_if:NT \g_@@_last_col_found_bool
3802   { \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`.

```

3803   \bool_if:NT \l_@@_last_col_without_value_bool
3804   { \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.

```

3805   \bool_if:NT \l_@@_last_row_without_value_bool
3806   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3807   \tl_gput_right:Ne \g_@@_aux_tl
3808   {
3809     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3810     {
3811       \int_use:N \l_@@_first_row_int ,
3812       \int_use:N \c@iRow ,
3813       \int_use:N \g_@@_row_total_int ,
3814       \int_use:N \l_@@_first_col_int ,
3815       \int_use:N \c@jCol ,
3816       \int_use:N \g_@@_col_total_int
3817     }
3818   }

```

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`).

```

3819 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3820 {
3821   \tl_gput_right:Ne \g_@@_aux_tl
3822   {
3823     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3824     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3825   }
3826 }
3827 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3828 {
3829   \tl_gput_right:Ne \g_@@_aux_tl
3830   {
3831     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3832     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3833     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3834     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3835   }
3836 }

```

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

```

3837 \@@_create_diag_nodes:

```

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

```

3838 \pgfpicture
3839 \int_step_inline:nn \c@iRow
3840 {
3841   \pgfnodealias
3842   { \@@_env: - ##1 - last }
3843   { \@@_env: - ##1 - \int_use:N \c@jCol }
3844 }
3845 \int_step_inline:nn \c@jCol
3846 {
3847   \pgfnodealias
3848   { \@@_env: - last - ##1 }
3849   { \@@_env: - \int_use:N \c@iRow - ##1 }
3850 }
3851 \str_if_empty:NF \l_@@_name_str
3852 {
3853   \int_step_inline:nn \c@iRow
3854   {
3855     \pgfnodealias
3856     { \l_@@_name_str - ##1 - last }
3857     { \@@_env: - ##1 - \int_use:N \c@jCol }
3858   }
3859   \int_step_inline:nn \c@jCol
3860   {
3861     \pgfnodealias
3862     { \l_@@_name_str - last - ##1 }
3863     { \@@_env: - \int_use:N \c@iRow - ##1 }
3864   }
3865 }
3866 \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.

```

3867 \bool_if:NT \l_@@_parallelize_diags_bool

```

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

```

3868 {
3869   \int_gzero_new:N \g_@@_ddots_int
3870   \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.

```

3871   \dim_gzero_new:N \g_@@_delta_x_one_dim
3872   \dim_gzero_new:N \g_@@_delta_y_one_dim
3873   \dim_gzero_new:N \g_@@_delta_x_two_dim
3874   \dim_gzero_new:N \g_@@_delta_y_two_dim
3875 }

```

```

3876 \int_zero_new:N \l_@@_initial_i_int
3877 \int_zero_new:N \l_@@_initial_j_int
3878 \int_zero_new:N \l_@@_final_i_int
3879 \int_zero_new:N \l_@@_final_j_int
3880 \bool_set_false:N \l_@@_initial_open_bool
3881 \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.

```

3882 \bool_if:NT \l_@@_small_bool
3883 {
3884   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3885   \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.

```

3886   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3887     { 0.6 \l_@@_xdots_shorten_start_dim }
3888   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3889     { 0.6 \l_@@_xdots_shorten_end_dim }
3890 }

```

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

```

3891 \@@_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.

```

3892 \clist_if_empty:NF \l_@@_corners_cells_clist
3893 {
3894   \bool_if:NTF \l_@@_no_cell_nodes_bool
3895     { \@@_error:n { corners~with~no~cell~nodes } }
3896     { \@@_compute_corners: }
3897 }

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3898 \@@_adjust_pos_of_blocks_seq:
3899 \@@_deal_with_rounded_corners:
3900 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3901 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3902 \IfPackageLoadedT { tikz }
3903 {
3904   \tikzset

```



```

3905     {
3906         every-picture / .style =
3907         {
3908             overlay ,
3909             remember~picture ,
3910             name~prefix = \@_env: -
3911         }
3912     }
3913 }
3914 \bool_if:NT \c_@@_recent_array_bool
3915 { \cs_set_eq:NN \ar@ialign \@_old_ar@ialign: }
3916 \cs_set_eq:NN \SubMatrix \@_SubMatrix
3917 \cs_set_eq:NN \UnderBrace \@_UnderBrace
3918 \cs_set_eq:NN \OverBrace \@_OverBrace
3919 \cs_set_eq:NN \ShowCellNames \@_ShowCellNames
3920 \cs_set_eq:NN \TikzEveryCell \@_TikzEveryCell
3921 \cs_set_eq:NN \line \@_line
3922 \g_@@_pre_code_after_tl
3923 \tl_gclear:N \g_@@_pre_code_after_tl

```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\CodeAfter` to be *no-op* now.

```

3924 \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).

```

3925 \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`).

```

3926 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
3927 { \@_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:`.

```

3928 \bool_set_true:N \l_@@_in_code_after_bool
3929 \exp_last_unbraced:No \@_CodeAfter_keys: \g_nicematrix_code_after_tl
3930 \scan_stop:
3931 \tl_gclear:N \g_nicematrix_code_after_tl
3932 \group_end:

```

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

```

3933 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@_clear_rowlistcolors_seq: }
3934 \tl_if_empty:NF \g_@@_pre_code_before_tl
3935 {
3936     \tl_gput_right:Ne \g_@@_aux_tl
3937     {
3938         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3939         { \exp_not:o \g_@@_pre_code_before_tl }
3940     }
3941     \tl_gclear:N \g_@@_pre_code_before_tl
3942 }
3943 \tl_if_empty:NF \g_nicematrix_code_before_tl
3944 {
3945     \tl_gput_right:Ne \g_@@_aux_tl
3946     {
3947         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3948         { \exp_not:o \g_nicematrix_code_before_tl }

```

```

3949     }
3950     \tl_gclear:N \g_nicematrix_code_before_tl
3951 }

```

```

3952 \str_gclear:N \g_@@_name_env_str
3953 \@@_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`.

```

3954 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3955 }

```

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`.

```

3956 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3957 { \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).

```

3958 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3959 {
3960     \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3961     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3962 }

```

The following command must *not* be protected.

```

3963 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3964 {
3965     { #1 }
3966     { #2 }
3967     {
3968         \int_compare:nNnTF { #3 } > { 98 }
3969         { \int_use:N \c@iRow }
3970         { #3 }
3971     }
3972     {
3973         \int_compare:nNnTF { #4 } > { 98 }
3974         { \int_use:N \c@jCol }
3975         { #4 }
3976     }
3977     { #5 }
3978 }

```

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).

```

3979 \hook_gput_code:nnn { begindocument } { . }
3980 {
3981     \cs_new_protected:Npe \@@_draw_dotted_lines:
3982     {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

3983         \c_@@_pgfortikzpicture_tl
3984         \@@_draw_dotted_lines_i:
3985         \c_@@_endpgfortikzpicture_tl
3986     }
3987 }

```

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

```

3988 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3989 {
3990     \pgfrememberpicturepositiononpagetrue
3991     \pgf@relevantforpicturesizefalse
3992     \g_@@_HVDotsfor_lines_tl
3993     \g_@@_VDots_lines_tl
3994     \g_@@_Ddots_lines_tl
3995     \g_@@_Iddots_lines_tl
3996     \g_@@_Cdots_lines_tl
3997     \g_@@_Ldots_lines_tl
3998 }

3999 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4000 {
4001     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4002     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4003 }

```

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

```

4004 \pgfdeclareshape { @@_diag_node }
4005 {
4006     \savedanchor { \five }
4007     {
4008         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4009         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4010     }
4011     \anchor { 5 } { \five }
4012     \anchor { center } { \pgfpointorigin }
4013     \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4014     \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4015     \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
4016     \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4017     \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4018     \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4019     \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4020     \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
4021     \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4022     \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4023 }

```

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).

```

4024 \cs_new_protected:Npn \@@_create_diag_nodes:
4025 {
4026     \pgfpicture
4027     \pgfrememberpicturepositiononpagetrue
4028     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4029     {
4030         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4031         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4032         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4033         \dim_set_eq:NN \l_tmpb_dim \pgf@y

```

```

4034 \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4035 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4036 \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4037 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4038 \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.

```

4039 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4040 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4041 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4042 \str_if_empty:NF \l_@@_name_str
4043 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4044 }

```

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

```

4045 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4046 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4047 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4048 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4049 \pgfcoordinate
4050 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4051 \pgfnodealias
4052 { \@@_env: - last }
4053 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4054 \str_if_empty:NF \l_@@_name_str
4055 {
4056 \pgfnodealias
4057 { \l_@@_name_str - \int_use:N \l_tmpa_int }
4058 { \@@_env: - \int_use:N \l_tmpa_int }
4059 \pgfnodealias
4060 { \l_@@_name_str - last }
4061 { \@@_env: - last }
4062 }
4063 \endpgfpicture
4064 }

```

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.

```
4065 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4066 {
```

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

```
4067 \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4068 \int_set:Nn \l_@@_initial_i_int { #1 }
4069 \int_set:Nn \l_@@_initial_j_int { #2 }
4070 \int_set:Nn \l_@@_final_i_int { #1 }
4071 \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.

```
4072 \bool_set_false:N \l_@@_stop_loop_bool
4073 \bool_do_until:Nn \l_@@_stop_loop_bool
4074 {
4075   \int_add:Nn \l_@@_final_i_int { #3 }
4076   \int_add:Nn \l_@@_final_j_int { #4 }
4077   \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.

```
4078   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4079     \if_int_compare:w #3 = \c_one_int
4080       \bool_set_true:N \l_@@_final_open_bool
4081     \else:
4082       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4083         \bool_set_true:N \l_@@_final_open_bool
4084       \fi:
4085     \fi:
4086   \else:
4087     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4088       \if_int_compare:w #4 = -1
4089         \bool_set_true:N \l_@@_final_open_bool
4090       \fi:
4091     \else:
4092       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4093         \if_int_compare:w #4 = \c_one_int
4094           \bool_set_true:N \l_@@_final_open_bool
4095         \fi:
4096       \fi:
4097     \fi:
4098   \fi:
4099   \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.

```
4100   {
```

We do a step backwards.

```
4101     \int_sub:Nn \l_@@_final_i_int { #3 }
4102     \int_sub:Nn \l_@@_final_j_int { #4 }
4103     \bool_set_true:N \l_@@_stop_loop_bool
4104   }
```

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`.

```

4105     {
4106         \cs_if_exist:cTF
4107         {
4108             @@ _ dotted _
4109             \int_use:N \l_@@_final_i_int -
4110             \int_use:N \l_@@_final_j_int
4111         }
4112         {
4113             \int_sub:Nn \l_@@_final_i_int { #3 }
4114             \int_sub:Nn \l_@@_final_j_int { #4 }
4115             \bool_set_true:N \l_@@_final_open_bool
4116             \bool_set_true:N \l_@@_stop_loop_bool
4117         }
4118         {
4119             \cs_if_exist:cTF
4120             {
4121                 pgf @ sh @ ns @ \@@_env:
4122                 - \int_use:N \l_@@_final_i_int
4123                 - \int_use:N \l_@@_final_j_int
4124             }
4125             { \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.

```

4126         {
4127             \cs_set_nopar:cpn
4128             {
4129                 @@ _ dotted _
4130                 \int_use:N \l_@@_final_i_int -
4131                 \int_use:N \l_@@_final_j_int
4132             }
4133             { }
4134         }
4135     }
4136 }
4137

```

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

```

4138     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4139     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4140     \bool_do_until:Nn \l_@@_stop_loop_bool
4141     {
4142         \int_sub:Nn \l_@@_initial_i_int { #3 }
4143         \int_sub:Nn \l_@@_initial_j_int { #4 }
4144         \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.

```

4145     \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4146     \if_int_compare:w #3 = \c_one_int
4147     \bool_set_true:N \l_@@_initial_open_bool
4148     \else:

```

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

```

4149         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4150         \bool_set_true:N \l_@@_initial_open_bool
4151     \fi:
4152 \fi:
4153 \else:
4154     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4155     \if_int_compare:w #4 = \c_one_int
4156     \bool_set_true:N \l_@@_initial_open_bool
4157     \fi:
4158 \else:
4159     \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4160     \if_int_compare:w #4 = -1
4161     \bool_set_true:N \l_@@_initial_open_bool
4162     \fi:
4163 \fi:
4164 \fi:
4165 \fi:
4166 \bool_if:NTF \l_@@_initial_open_bool
4167 {
4168     \int_add:Nn \l_@@_initial_i_int { #3 }
4169     \int_add:Nn \l_@@_initial_j_int { #4 }
4170     \bool_set_true:N \l_@@_stop_loop_bool
4171 }
4172 {
4173     \cs_if_exist:cTF
4174     {
4175         @@ _ dotted _
4176         \int_use:N \l_@@_initial_i_int -
4177         \int_use:N \l_@@_initial_j_int
4178     }
4179     {
4180         \int_add:Nn \l_@@_initial_i_int { #3 }
4181         \int_add:Nn \l_@@_initial_j_int { #4 }
4182         \bool_set_true:N \l_@@_initial_open_bool
4183         \bool_set_true:N \l_@@_stop_loop_bool
4184     }
4185     {
4186         \cs_if_exist:cTF
4187         {
4188             pgf @ sh @ ns @ \@@_env:
4189             - \int_use:N \l_@@_initial_i_int
4190             - \int_use:N \l_@@_initial_j_int
4191         }
4192         { \bool_set_true:N \l_@@_stop_loop_bool }
4193         {
4194             \cs_set_nopar:cpn
4195             {
4196                 @@ _ dotted _
4197                 \int_use:N \l_@@_initial_i_int -
4198                 \int_use:N \l_@@_initial_j_int
4199             }
4200             { }
4201         }
4202     }
4203 }
4204 }
```

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.

```

4205 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4206 {
4207     { \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`.

```

4208      { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4209      { \int_use:N \l_@@_final_i_int }
4210      { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4211      { } % for the name of the block
4212    }
4213  }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analysis 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.

```

4214 \cs_new_protected:Npn \@@_open_shorten:
4215 {
4216   \bool_if:NT \l_@@_initial_open_bool
4217   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4218   \bool_if:NT \l_@@_final_open_bool
4219   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4220 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it's only the whole array (excepted exterior rows and columns).

```

4221 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4222 {
4223   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4224   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4225   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4226   \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`.

```

4227   \seq_if_empty:NF \g_@@_submatrix_seq
4228   {
4229     \seq_map_inline:Nn \g_@@_submatrix_seq
4230     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4231   }
4232 }

```

`#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 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.

```

4233 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4234 {
4235   \if_int_compare:w #3 > #1
4236   \else:
4237     \if_int_compare:w #1 > #5
4238     \else:
4239       \if_int_compare:w #4 > #2
4240       \else:
4241         \if_int_compare:w #2 > #6
4242         \else:
4243           \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4244           \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4245           \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4246           \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4247         \fi:
4248       \fi:
4249     \fi:
4250   \fi:
4251 }

4252 \cs_new_protected:Npn \@@_set_initial_coords:
4253 {
4254   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4255   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4256 }
4257 \cs_new_protected:Npn \@@_set_final_coords:
4258 {
4259   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4260   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4261 }
4262 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4263 {
4264   \pgfpointanchor
4265   {
4266     \@@_env:
4267     - \int_use:N \l_@@_initial_i_int
4268     - \int_use:N \l_@@_initial_j_int
4269   }
4270   { #1 }
4271   \@@_set_initial_coords:
4272 }
4273 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4274 {
4275   \pgfpointanchor
4276   {
4277     \@@_env:
4278     - \int_use:N \l_@@_final_i_int
4279     - \int_use:N \l_@@_final_j_int
4280   }
4281   { #1 }
4282   \@@_set_final_coords:
4283 }

4284 \cs_new_protected:Npn \@@_open_x_initial_dim:
4285 {
4286   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4287   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4288   {
4289     \cs_if_exist:cT

```

```

4290 { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4291 {
4292   \pgfpointanchor
4293   { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4294   { west }
4295   \dim_set:Nn \l_@@_x_initial_dim
4296   { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4297 }
4298 }

```

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

```

4299 \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4300 {
4301   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4302   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4303   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4304 }
4305 }
4306 \cs_new_protected:Npn \@@_open_x_final_dim:
4307 {
4308   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4309   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4310   {
4311     \cs_if_exist:cT
4312     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4313     {
4314       \pgfpointanchor
4315       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4316       { east }
4317       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4318       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4319     }
4320   }

```

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

```

4321 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4322 {
4323   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4324   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4325   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4326 }
4327 }

```

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.

```

4328 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4329 {
4330   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4331   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4332   {
4333     \@@_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.

```

4334 \group_begin:
4335   \@@_open_shorten:
4336   \int_if_zero:nTF { #1 }
4337   { \color { nicematrix-first-row } }
4338   {

```

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.

```

4339 \int_compare:nNnT { #1 } = \l_@@_last_row_int

```

```

4340         { \color { nicematrix-last-row } }
4341     }
4342     \keys_set:nn { nicematrix / xdots } { #3 }
4343     \@@_color:o \l_@@_xdots_color_tl
4344     \@@_actually_draw_Ldots:
4345     \group_end:
4346 }
4347 }

```

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`.

```

4348 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4349 {
4350     \bool_if:NTF \l_@@_initial_open_bool
4351     {
4352         \@@_open_x_initial_dim:
4353         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4354         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4355     }
4356     { \@@_set_initial_coords_from_anchor:n { base~east } }
4357     \bool_if:NTF \l_@@_final_open_bool
4358     {
4359         \@@_open_x_final_dim:
4360         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4361         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4362     }
4363     { \@@_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.

```

4364     \bool_lazy_all:nTF
4365     {
4366         \l_@@_initial_open_bool
4367         \l_@@_final_open_bool
4368         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4369     }
4370     {
4371         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4372         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4373     }

```

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 (?).

```

4374     {
4375         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4376         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4377     }
4378     \@@_draw_line:
4379 }

```

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.

```

4380 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4381 {
4382   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4383   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4384   {
4385     \@@_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.

```

4386     \group_begin:
4387     \@@_open_shorten:
4388     \int_if_zero:nTF { #1 }
4389     { \color { nicematrix-first-row } }
4390     {

```

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.

```

4391         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4392         { \color { nicematrix-last-row } }
4393     }
4394     \keys_set:nn { nicematrix / xdots } { #3 }
4395     \@@_color:o \l_@@_xdots_color_tl
4396     \@@_actually_draw_Cdots:
4397   \group_end:
4398 }
4399 }

```

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`.

```

4400 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4401 {
4402   \bool_if:NTF \l_@@_initial_open_bool
4403   { \@@_open_x_initial_dim: }
4404   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4405   \bool_if:NTF \l_@@_final_open_bool
4406   { \@@_open_x_final_dim: }
4407   { \@@_set_final_coords_from_anchor:n { mid-west } }
4408   \bool_lazy_and:nnTF
4409   \l_@@_initial_open_bool
4410   \l_@@_final_open_bool
4411   {
4412     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4413     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4414     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4415     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4416     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4417   }
4418   {
4419     \bool_if:NT \l_@@_initial_open_bool
4420     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4421     \bool_if:NT \l_@@_final_open_bool
4422     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }

```

```

4423     }
4424     \@@_draw_line:
4425 }

4426 \cs_new_protected:Npn \@@_open_y_initial_dim:
4427 {
4428     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4429     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4430     {
4431         \cs_if_exist:cT
4432         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4433         {
4434             \pgfpointanchor
4435             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4436             { north }
4437             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim
4438             { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4439         }
4440     }
4441     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4442     {
4443         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4444         \dim_set:Nn \l_@@_y_initial_dim
4445         {
4446             \fp_to_dim:n
4447             {
4448                 \pgf@y
4449                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4450             }
4451         }
4452     }
4453 }

4454 \cs_new_protected:Npn \@@_open_y_final_dim:
4455 {
4456     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4457     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4458     {
4459         \cs_if_exist:cT
4460         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4461         {
4462             \pgfpointanchor
4463             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4464             { south }
4465             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4466             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4467         }
4468     }
4469     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4470     {
4471         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4472         \dim_set:Nn \l_@@_y_final_dim
4473         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4474     }
4475 }

```

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.

```

4476 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4477 {
4478     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4479     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4480     {
4481         \@@_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.

```

4482     \group_begin:
4483     \@@_open_shorten:
4484     \int_if_zero:nTF { #2 }
4485     { \color { nicematrix-first-col } }
4486     {
4487         \int_compare:nNtT { #2 } = \l_@@_last_col_int
4488         { \color { nicematrix-last-col } }
4489     }
4490     \keys_set:nn { nicematrix / xdots } { #3 }
4491     \@@_color:o \l_@@_xdots_color_tl
4492     \@@_actually_draw_Vdots:
4493 \group_end:
4494 }
4495 }
```

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`.

```

4496 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4497 {
```

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

```

4498     \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.

```

4499     {
4500         \@@_open_y_initial_dim:
4501         \@@_open_y_final_dim:
4502         \int_if_zero:nTF \l_@@_initial_j_int
```

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

```

4503     {
4504         \@@_qpoint:n { col - 1 }
4505         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4506         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4507         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4508         \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4509     }
4510     {
4511         \bool_lazy_and:nnTF
4512         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4513         { \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”.

```

4514     {
4515         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4516         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4517         \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4518         \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4519         \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4520     }
```

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

```

4521     {
4522         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4523         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4524         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4525         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4526     }
4527 }
4528 }

```

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.

```

4529 {
4530     \bool_set_false:N \l_tmpa_bool
4531     \bool_if:NF \l_@@_initial_open_bool
4532     {
4533         \bool_if:NF \l_@@_final_open_bool
4534         {
4535             \@@_set_initial_coords_from_anchor:n { south-west }
4536             \@@_set_final_coords_from_anchor:n { north-west }
4537             \bool_set:Nn \l_tmpa_bool
4538                 { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4539         }
4540     }

```

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

```

4541     \bool_if:NTF \l_@@_initial_open_bool
4542     {
4543         \@@_open_y_initial_dim:
4544         \@@_set_final_coords_from_anchor:n { north }
4545         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4546     }
4547     {
4548         \@@_set_initial_coords_from_anchor:n { south }
4549         \bool_if:NTF \l_@@_final_open_bool
4550         \@@_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.

```

4551     {
4552         \@@_set_final_coords_from_anchor:n { north }
4553         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4554         {
4555             \dim_set:Nn \l_@@_x_initial_dim
4556             {
4557                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4558                 \l_@@_x_initial_dim \l_@@_x_final_dim
4559             }
4560         }
4561     }
4562 }
4563 }
4564 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4565 \@@_draw_line:
4566 }

```

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.

```

4567 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4568 {

```

```

4569 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4570 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4571 {
4572 \@@_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.

```

4573 \group_begin:
4574 \@@_open_shorten:
4575 \keys_set:nn { nicematrix / xdots } { #3 }
4576 \@@_color:o \l_@@_xdots_color_tl
4577 \@@_actually_draw_Ddots:
4578 \group_end:
4579 }
4580 }

```

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.`

```

4581 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4582 {
4583 \bool_if:NTF \l_@@_initial_open_bool
4584 {
4585 \@@_open_y_initial_dim:
4586 \@@_open_x_initial_dim:
4587 }
4588 { \@@_set_initial_coords_from_anchor:n { south-east } }
4589 \bool_if:NTF \l_@@_final_open_bool
4590 {
4591 \@@_open_x_final_dim:
4592 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4593 }
4594 { \@@_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.

```

4595 \bool_if:NT \l_@@_parallelize_diags_bool
4596 {
4597 \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).

```

4598 \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.

```

4599 {
4600 \dim_gset:Nn \g_@@_delta_x_one_dim
4601 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4602 \dim_gset:Nn \g_@@_delta_y_one_dim
4603 { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4604 }

```


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`.

```

4605     {
4606         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4607         {
4608             \dim_set:Nn \l_@@_y_final_dim
4609             {
4610                 \l_@@_y_initial_dim +
4611                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4612                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4613             }
4614         }
4615     }
4616 }
4617 \@@_draw_line:
4618 }

```

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.

```

4619 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4620 {
4621     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4622     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4623     {
4624         \@@_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.

```

4625     \group_begin:
4626     \@@_open_shorten:
4627     \keys_set:nn { nicematrix / xdots } { #3 }
4628     \@@_color:o \l_@@_xdots_color_tl
4629     \@@_actually_draw_Iddots:
4630     \group_end:
4631 }
4632 }

```

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`.

```

4633 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4634 {
4635     \bool_if:NTF \l_@@_initial_open_bool
4636     {
4637         \@@_open_y_initial_dim:
4638         \@@_open_x_initial_dim:
4639     }
4640     { \@@_set_initial_coords_from_anchor:n { south~west } }
4641     \bool_if:NTF \l_@@_final_open_bool
4642     {
4643         \@@_open_y_final_dim:
4644         \@@_open_x_final_dim:

```

```

4645     }
4646     { \@@_set_final_coords_from_anchor:n { north-east } }
4647     \bool_if:NT \l_@@_parallelize_diags_bool
4648     {
4649         \int_gincr:N \g_@@_iddots_int
4650         \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4651         {
4652             \dim_gset:Nn \g_@@_delta_x_two_dim
4653             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4654             \dim_gset:Nn \g_@@_delta_y_two_dim
4655             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4656         }
4657         {
4658             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4659             {
4660                 \dim_set:Nn \l_@@_y_final_dim
4661                 {
4662                     \l_@@_y_initial_dim +
4663                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4664                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4665                 }
4666             }
4667         }
4668     }
4669     \@@_draw_line:
4670 }

```

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`

```

4671 \cs_new_protected:Npn \@@_draw_line:
4672 {
4673     \pgfrememberpicturepositiononpagetrue
4674     \pgf@relevantforpicturesizefalse
4675     \bool_lazy_or:nnTF
4676     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4677     \l_@@_dotted_bool
4678     \@@_draw_standard_dotted_line:
4679     \@@_draw_unstandard_dotted_line:
4680 }

```

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.

```

4681 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4682 {

```

```

4683 \begin { scope }
4684 \@@_draw_unstandard_dotted_line:o
4685 { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4686 }

```

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.

```

4687 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4688 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4689 {
4690 \@@_draw_unstandard_dotted_line:nooo
4691 { #1 }
4692 \l_@@_xdots_up_tl
4693 \l_@@_xdots_down_tl
4694 \l_@@_xdots_middle_tl
4695 }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4696 \hook_gput_code:nnn { begindocument } { . }
4697 {
4698 \IfPackageLoadedT { tikz }
4699 {
4700 \tikzset
4701 {
4702 @@_node_above / .style = { sloped , above } ,
4703 @@_node_below / .style = { sloped , below } ,
4704 @@_node_middle / .style =
4705 {
4706 sloped ,
4707 inner~sep = \c_@@_innersep_middle_dim
4708 }
4709 }
4710 }
4711 }

4712 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4713 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4714 {

```

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.

```

4715 \dim_zero_new:N \l_@@_l_dim
4716 \dim_set:Nn \l_@@_l_dim
4717 {
4718 \fp_to_dim:n
4719 {
4720 sqrt
4721 (
4722 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4723 +
4724 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4725 )
4726 }
4727 }

```

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.

```

4728 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4729 {
4730   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4731   \@@_draw_unstandard_dotted_line_i:
4732 }

```

If the key `xdots/horizontal-labels` has been used.

```

4733 \bool_if:NT \l_@@_xdots_h_labels_bool
4734 {
4735   \tikzset
4736   {
4737     @@_node_above / .style = { auto = left } ,
4738     @@_node_below / .style = { auto = right } ,
4739     @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4740   }
4741 }
4742 \tl_if_empty:nF { #4 }
4743 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4744 \draw
4745 [ #1 ]
4746 ( \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).

```

4747 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4748 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4749 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4750 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4751 \end { scope }
4752 }
4753 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4754 {
4755   \dim_set:Nn \l_tmpa_dim
4756   {
4757     \l_@@_x_initial_dim
4758     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4759     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4760   }
4761   \dim_set:Nn \l_tmpb_dim
4762   {
4763     \l_@@_y_initial_dim
4764     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4765     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4766   }
4767   \dim_set:Nn \l_@@_tmpc_dim
4768   {
4769     \l_@@_x_final_dim
4770     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4771     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4772   }
4773   \dim_set:Nn \l_@@_tmpd_dim
4774   {
4775     \l_@@_y_final_dim
4776     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4777     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4778   }
4779   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4780   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4781   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim

```

```

4782 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4783 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4784 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4785 {
4786   \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.

```

4787   \dim_zero_new:N \l_@@_l_dim
4788   \dim_set:Nn \l_@@_l_dim
4789   {
4790     \fp_to_dim:n
4791     {
4792       sqrt
4793       (
4794         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4795         +
4796         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4797       )
4798     }
4799   }

```

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.

```

4800   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4801   {
4802     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4803     \@@_draw_standard_dotted_line_i:
4804   }
4805   \group_end:
4806   \bool_lazy_all:nF
4807   {
4808     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4809     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4810     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4811   }
4812   \l_@@_labels_standard_dotted_line:
4813 }
4814 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4815 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4816 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4817   \int_set:Nn \l_tmpa_int
4818   {
4819     \dim_ratio:nn
4820     {
4821       \l_@@_l_dim
4822       - \l_@@_xdots_shorten_start_dim
4823       - \l_@@_xdots_shorten_end_dim
4824     }
4825     \l_@@_xdots_inter_dim
4826   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4827   \dim_set:Nn \l_tmpa_dim

```

```

4828 {
4829   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4830   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4831 }
4832 \dim_set:Nn \l_tmpb_dim
4833 {
4834   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4835   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4836 }

```

In the loop over the dots, the dimensions `\l_@@_x_initial_dim` and `\l_@@_y_initial_dim` will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4837 \dim_gadd:Nn \l_@@_x_initial_dim
4838 {
4839   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4840   \dim_ratio:nn
4841   {
4842     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4843     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4844   }
4845   { 2 \l_@@_l_dim }
4846 }
4847 \dim_gadd:Nn \l_@@_y_initial_dim
4848 {
4849   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4850   \dim_ratio:nn
4851   {
4852     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4853     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4854   }
4855   { 2 \l_@@_l_dim }
4856 }
4857 \pgf@relevantforpicturesizefalse
4858 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4859 {
4860   \pgfpathcircle
4861   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4862   { \l_@@_xdots_radius_dim }
4863   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4864   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4865 }
4866 \pgfusepathqfill
4867 }

```

```

4868 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4869 {
4870   \pgfscope
4871   \pgftransformshift
4872   {
4873     \pgfpointlineattime { 0.5 }
4874     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4875     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4876   }
4877   \fp_set:Nn \l_tmpa_fp
4878   {
4879     atand
4880     (
4881       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4882       \l_@@_x_final_dim - \l_@@_x_initial_dim
4883     )
4884   }
4885   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4886   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }

```

```

4887 \tl_if_empty:NF \l_@@_xdots_middle_tl
4888 {
4889   \begin { pgfscope }
4890   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4891   \pgfnode
4892     { rectangle }
4893     { center }
4894     {
4895       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4896       {
4897         \c_math_toggle_token
4898         \scriptstyle \l_@@_xdots_middle_tl
4899         \c_math_toggle_token
4900       }
4901     }
4902     { }
4903     {
4904       \pgfsetfillcolor { white }
4905       \pgfusepath { fill }
4906     }
4907   \end { pgfscope }
4908 }
4909 \tl_if_empty:NF \l_@@_xdots_up_tl
4910 {
4911   \pgfnode
4912     { rectangle }
4913     { south }
4914     {
4915       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4916       {
4917         \c_math_toggle_token
4918         \scriptstyle \l_@@_xdots_up_tl
4919         \c_math_toggle_token
4920       }
4921     }
4922     { }
4923     { \pgfusepath { } }
4924 }
4925 \tl_if_empty:NF \l_@@_xdots_down_tl
4926 {
4927   \pgfnode
4928     { rectangle }
4929     { north }
4930     {
4931       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4932       {
4933         \c_math_toggle_token
4934         \scriptstyle \l_@@_xdots_down_tl
4935         \c_math_toggle_token
4936       }
4937     }
4938     { }
4939     { \pgfusepath { } }
4940 }
4941 \endpgfscope
4942 }

```

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.

```

4943 \hook_gput_code:nnn { begindocument } { . }
4944 {
4945   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { } { } { } }
4946   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4947   \cs_new_protected:Npn \@@_Ldots
4948     { \@@_collect_options:n { \@@_Ldots_i } }
4949   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4950     {
4951       \int_if_zero:nTF \c@jCol
4952       { \@@_error:nn { in~first~col } \Ldots }
4953       {
4954         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4955         { \@@_error:nn { in~last~col } \Ldots }
4956         {
4957           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4958           { #1 , down = #2 , up = #3 , middle = #4 }
4959         }
4960       }
4961       \bool_if:NF \l_@@_nullify_dots_bool
4962       { \phantom { \ensuremath { \@@_old_ldots } } }
4963       \bool_gset_true:N \g_@@_empty_cell_bool
4964     }

4965   \cs_new_protected:Npn \@@_Cdots
4966     { \@@_collect_options:n { \@@_Cdots_i } }
4967   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4968     {
4969       \int_if_zero:nTF \c@jCol
4970       { \@@_error:nn { in~first~col } \Cdots }
4971       {
4972         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4973         { \@@_error:nn { in~last~col } \Cdots }
4974         {
4975           \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4976           { #1 , down = #2 , up = #3 , middle = #4 }
4977         }
4978       }
4979       \bool_if:NF \l_@@_nullify_dots_bool
4980       { \phantom { \ensuremath { \@@_old_cdots } } }
4981       \bool_gset_true:N \g_@@_empty_cell_bool
4982     }

4983   \cs_new_protected:Npn \@@_Vdots
4984     { \@@_collect_options:n { \@@_Vdots_i } }
4985   \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4986     {
4987       \int_if_zero:nTF \c@iRow
4988       { \@@_error:nn { in~first~row } \Vdots }
4989       {

```



```

4990         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4991         { \@@_error:nn { in~last~row } \Vdots }
4992         {
4993             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4994             { #1 , down = #2 , up = #3 , middle = #4 }
4995         }
4996     }
4997     \bool_if:NF \l_@@_nullify_dots_bool
4998     { \phantom { \ensuremath { \@@_old_vdots } } }
4999     \bool_gset_true:N \g_@@_empty_cell_bool
5000 }

5001 \cs_new_protected:Npn \@@_Ddots
5002 { \@@_collect_options:n { \@@_Ddots_i } }
5003 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5004 {
5005     \int_case:nnF \c@iRow
5006     {
5007         0 { \@@_error:nn { in~first~row } \Ddots }
5008         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5009     }
5010     {
5011         \int_case:nnF \c@jCol
5012         {
5013             0 { \@@_error:nn { in~first~col } \Ddots }
5014             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5015         }
5016         {
5017             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5018             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5019             { #1 , down = #2 , up = #3 , middle = #4 }
5020         }
5021     }
5022 }
5023 \bool_if:NF \l_@@_nullify_dots_bool
5024 { \phantom { \ensuremath { \@@_old_ddots } } }
5025 \bool_gset_true:N \g_@@_empty_cell_bool
5026 }

5027 \cs_new_protected:Npn \@@_Iddots
5028 { \@@_collect_options:n { \@@_Iddots_i } }
5029 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5030 {
5031     \int_case:nnF \c@iRow
5032     {
5033         0 { \@@_error:nn { in~first~row } \Iddots }
5034         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5035     }
5036     {
5037         \int_case:nnF \c@jCol
5038         {
5039             0 { \@@_error:nn { in~first~col } \Iddots }
5040             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5041         }
5042         {
5043             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5044             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5045             { #1 , down = #2 , up = #3 , middle = #4 }
5046         }
5047     }
5048     \bool_if:NF \l_@@_nullify_dots_bool
5049     { \phantom { \ensuremath { \@@_old_iddots } } }

```

```

5050     \bool_gset_true:N \g_@@_empty_cell_bool
5051   }
5052 }

```

End of the `\AddToHook`.

Despite its name, the following set of keys will be used for `\Ddots` but also for `\Iddots`.

```

5053 \keys_define:nn { nicematrix / Ddots }
5054 {
5055   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5056   draw-first .default:n = true ,
5057   draw-first .value_forbidden:n = true
5058 }

```

The command `\@@_Hspace:` will be linked to `\hspace` in `{NiceArray}`.

```

5059 \cs_new_protected:Npn \@@_Hspace:
5060 {
5061   \bool_gset_true:N \g_@@_empty_cell_bool
5062   \hspace
5063 }

```

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`.

```

5064 \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`.

```

5065 \cs_new:Npn \@@_Hdotsfor:
5066 {
5067   \bool_lazy_and:nnTF
5068     { \int_if_zero_p:n \c@jCol }
5069     { \int_if_zero_p:n \l_@@_first_col_int }
5070   {
5071     \bool_if:NTF \g_@@_after_col_zero_bool
5072     {
5073       \multicolumn { 1 } { c } { }
5074       \@@_Hdotsfor_i
5075     }
5076     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5077   }
5078   {
5079     \multicolumn { 1 } { c } { }
5080     \@@_Hdotsfor_i
5081   }
5082 }

```

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:`).

```

5083 \hook_gput_code:nnn { begindocument } { . }
5084 {
5085   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5086   \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.

```

5087   \cs_new_protected:Npn \@@_Hdotsfor_i
5088     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5089   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5090   {

```

```

5091 \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5092 {
5093   \@@_Hdotsfor:nnnn
5094   { \int_use:N \c@iRow }
5095   { \int_use:N \c@jCol }
5096   { #2 }
5097   {
5098     #1 , #3 ,
5099     down = \exp_not:n { #4 } ,
5100     up = \exp_not:n { #5 } ,
5101     middle = \exp_not:n { #6 }
5102   }
5103 }
5104 \prg_replicate:nn { #2 - 1 }
5105 {
5106   &
5107   \multicolumn { 1 } { c } { }
5108   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5109 }
5110 }
5111 }

```

```

5112 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5113 {
5114   \bool_set_false:N \l_@@_initial_open_bool
5115   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5116 \int_set:Nn \l_@@_initial_i_int { #1 }
5117 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5118 \int_compare:nNnTF { #2 } = \c_one_int
5119 {
5120   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5121   \bool_set_true:N \l_@@_initial_open_bool
5122 }
5123 {
5124   \cs_if_exist:cTF
5125   {
5126     pgf @ sh @ ns @ \@@_env:
5127     - \int_use:N \l_@@_initial_i_int
5128     - \int_eval:n { #2 - 1 }
5129   }
5130   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5131   {
5132     \int_set:Nn \l_@@_initial_j_int { #2 }
5133     \bool_set_true:N \l_@@_initial_open_bool
5134   }
5135 }
5136 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5137 {
5138   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5139   \bool_set_true:N \l_@@_final_open_bool
5140 }
5141 {
5142   \cs_if_exist:cTF
5143   {
5144     pgf @ sh @ ns @ \@@_env:
5145     - \int_use:N \l_@@_final_i_int
5146     - \int_eval:n { #2 + #3 }
5147   }
5148   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5149   {

```

```

5150         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5151         \bool_set_true:N \l_@@_final_open_bool
5152     }
5153 }
5154 \group_begin:
5155 \@@_open_shorten:
5156 \int_if_zero:nTF { #1 }
5157 { \color { nicematrix-first-row } }
5158 {
5159     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5160     { \color { nicematrix-last-row } }
5161 }
5162
5163 \keys_set:nn { nicematrix / xdots } { #4 }
5164 \@@_color:o \l_@@_xdots_color_tl
5165 \@@_actually_draw_Ldots:
5166 \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).

```

5167     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5168     { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5169 }

5170 \hook_gput_code:nnn { begindocument } { . }
5171 {
5172     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5173     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5174     \cs_new_protected:Npn \@@_Vdotsfor:
5175     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5176     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5177     {
5178         \bool_gset_true:N \g_@@_empty_cell_bool
5179         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5180         {
5181             \@@_Vdotsfor:nnnn
5182             { \int_use:N \c@iRow }
5183             { \int_use:N \c@jCol }
5184             { #2 }
5185             {
5186                 #1 , #3 ,
5187                 down = \exp_not:n { #4 } ,
5188                 up = \exp_not:n { #5 } ,
5189                 middle = \exp_not:n { #6 }
5190             }
5191         }
5192     }
5193 }

5194 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5195 {
5196     \bool_set_false:N \l_@@_initial_open_bool
5197     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5198     \int_set:Nn \l_@@_initial_j_int { #2 }
5199     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5200 \int_compare:nNnTF { #1 } = \c_one_int
5201 {
5202   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5203   \bool_set_true:N \l_@@_initial_open_bool
5204 }
5205 {
5206   \cs_if_exist:cTF
5207   {
5208     pgf @ sh @ ns @ \@@_env:
5209     - \int_eval:n { #1 - 1 }
5210     - \int_use:N \l_@@_initial_j_int
5211   }
5212   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5213   {
5214     \int_set:Nn \l_@@_initial_i_int { #1 }
5215     \bool_set_true:N \l_@@_initial_open_bool
5216   }
5217 }
5218 \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5219 {
5220   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5221   \bool_set_true:N \l_@@_final_open_bool
5222 }
5223 {
5224   \cs_if_exist:cTF
5225   {
5226     pgf @ sh @ ns @ \@@_env:
5227     - \int_eval:n { #1 + #3 }
5228     - \int_use:N \l_@@_final_j_int
5229   }
5230   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5231   {
5232     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5233     \bool_set_true:N \l_@@_final_open_bool
5234   }
5235 }
5236 \group_begin:
5237 \@@_open_shorten:
5238 \int_if_zero:nTF { #2 }
5239 { \color { nicematrix-first-col } }
5240 {
5241   \int_compare:nNnT { #2 } = \g_@@_col_total_int
5242   { \color { nicematrix-last-col } }
5243 }
5244 \keys_set:nn { nicematrix / xdots } { #4 }
5245 \@@_color:o \l_@@_xdots_color_tl
5246 \@@_actually_draw_Vdots:
5247 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

5248 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5249 { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5250 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5251 \NewDocumentCommand \@@_rotate: { 0 { } }
5252 {

```

```

5253 \peek_remove_spaces:n
5254 {
5255     \bool_gset_true:N \g_@@_rotate_bool
5256     \keys_set:nn { nicematrix / rotate } { #1 }
5257 }
5258 }

5259 \keys_define:nn { nicematrix / rotate }
5260 {
5261     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5262     c .value_forbidden:n = true ,
5263     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5264 }

```

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

```

5265 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5266 {
5267     \tl_if_empty:nTF { #2 }
5268     { #1 }
5269     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5270 }
5271 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5272 { \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).

```

5273 \hook_gput_code:nnn { begindocument } { . }
5274 {
5275     \cs_set_nopar:Npn \l_@@_argspec_tl
5276     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5277     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5278     \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5279     {
5280         \group_begin:
5281         \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5282         \@@_color:o \l_@@_xdots_color_tl
5283         \use:e
5284         {
5285             \@@_line_i:nn
5286             { \@@_double_int_eval:n #2 - \q_stop }
5287             { \@@_double_int_eval:n #3 - \q_stop }

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5288     }
5289     \group_end:
5290 }
5291 }

5292 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5293 {
5294     \bool_set_false:N \l_@@_initial_open_bool
5295     \bool_set_false:N \l_@@_final_open_bool
5296     \bool_lazy_or:nnTF
5297     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5298     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5299     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5300     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5301 }

5302 \hook_gput_code:nnn { begindocument } { . }
5303 {
5304     \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5305     {

```

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:`.

```

5306         \c_@@_pgfortikzpicture_tl
5307         \@@_draw_line_iii:nn { #1 } { #2 }
5308         \c_@@_endpgfortikzpicture_tl
5309     }
5310 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5311 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5312 {
5313     \pgfrememberpicturepositiononpagetrue
5314     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5315     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5316     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5317     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5318     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5319     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5320     \@@_draw_line:
5321 }

```

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots` don’t use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

20 The command `\RowStyle`

`\g_@@_row_style_tl` may contain several instructions of the form:

```
\@@_if_row_less_than:nn { number } { instructions }
```

Then, `\g_@@_row_style_tl` will be inserted in all the cells of the array (and also in both components of a `\diagbox` in a cell of in a mono-row block).

The test `\@@_if_row_less_then:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_then:nn` is *not* protected.

#1 is the first row *after* the scope of the instructions in #2

```
5322 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5323 { \int_compare:nNtT { \c@iRow } < { #1 } { #2 } }
```

\@@_put_in_row_style will be used several times in \RowStyle.

```
5324 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5325 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5326 {
5327   \tl_gput_right:Ne \g_@@_row_style_tl
5328   {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```
5329     \exp_not:N
5330     \@@_if_row_less_than:nn
5331     { \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).

```
5332     { \exp_not:n { #1 } \scan_stop: }
5333   }
5334 }
```

```
5335 \keys_define:nn { nicematrix / RowStyle }
5336 {
5337   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5338   cell-space-top-limit .value_required:n = true ,
5339   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5340   cell-space-bottom-limit .value_required:n = true ,
5341   cell-space-limits .meta:n =
5342   {
5343     cell-space-top-limit = #1 ,
5344     cell-space-bottom-limit = #1 ,
5345   } ,
5346   color .tl_set:N = \l_@@_color_tl ,
5347   color .value_required:n = true ,
5348   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5349   bold .default:n = true ,
5350   nb-rows .code:n =
5351   { \str_if_eq:eeTF { #1 } { * }
5352     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5353     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5354   nb-rows .value_required:n = true ,
5355   fill .tl_set:N = \l_@@_fill_tl ,
5356   fill .value_required:n = true ,
5357   opacity .tl_set:N = \l_@@_opacity_tl ,
5358   opacity .value_required:n = true ,
5359   rowcolor .tl_set:N = \l_@@_fill_tl ,
5360   rowcolor .value_required:n = true ,
5361   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5362   rounded-corners .default:n = 4 pt ,
5363   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5364 }
```

```
5365 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5366 {
5367   \group_begin:
5368   \tl_clear:N \l_@@_fill_tl
5369   \tl_clear:N \l_@@_opacity_tl
5370   \tl_clear:N \l_@@_color_tl
5371   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5372   \dim_zero:N \l_@@_rounded_corners_dim
```



```

5373 \dim_zero:N \l_tmpa_dim
5374 \dim_zero:N \l_tmpb_dim
5375 \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` (of its alias `fill`) has been used.

```

5376 \tl_if_empty:NF \l_@@_fill_tl
5377 {
5378   \@@_add_opacity_to_fill:
5379   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5380   {

```

First, the case when the command `\RowStyle` is *not* issued in the first column of the array. In that case, the command applies to the end of the row in the row where the command `\RowStyle` is issued, but in the other whole rows, if the key `nb-rows` is used.

```

5381 \int_compare:nNnTF \c@jCol > \c_one_int
5382 {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row). The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5383 \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
5384 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5385 { \int_use:N \c@iRow - * }
5386 { \dim_use:N \l_@@_rounded_corners_dim }

```

Then, the other rows (if there are several rows).

```

5387 \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5388 { \@@_rounded_from_row:n { \c@iRow + 1 } }
5389 }

```

Now, directly all the rows in the case of a command `\RowStyle` issued in the first column of the array.

```

5390 { \@@_rounded_from_row:n { \c@iRow } }
5391 }
5392 }
5393 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5394 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5395 {
5396   \@@_put_in_row_style:e
5397   {
5398     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5399     {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5400 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5401 { \dim_use:N \l_tmpa_dim }
5402 }
5403 }
5404 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5405 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5406 {
5407   \@@_put_in_row_style:e
5408   {
5409     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5410     {
5411       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5412       { \dim_use:N \l_tmpb_dim }
5413     }
5414   }
5415 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5416   \tl_if_empty:NF \l_@@_color_tl
5417   {
5418     \@@_put_in_row_style:e
5419     {
5420       \mode_leave_vertical:
5421       \@@_color:n { \l_@@_color_tl }
5422     }
5423   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5424   \bool_if:NT \l_@@_bold_row_style_bool
5425   {
5426     \@@_put_in_row_style:n
5427     {
5428       \exp_not:n
5429       {
5430         \if_mode_math:
5431           \c_math_toggle_token
5432           \bfseries \boldmath
5433           \c_math_toggle_token
5434         \else:
5435           \bfseries \boldmath
5436         \fi:
5437       }
5438     }
5439   }
5440   \group_end:
5441   \g_@@_row_style_tl
5442   \ignorespaces
5443 }

```

The following commande must *not* be protected.

```

5444 \cs_new:Npn \@@_rounded_from_row:n #1
5445 {
5446   \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl

```

In the following code, the “`- 1`” is *not* a subtraction.

```

5447   { \int_eval:n { #1 } - 1 }
5448   {
5449     \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5450     - \exp_not:n { \int_use:N \c@jCol }
5451   }
5452   { \dim_use:N \l_@@_rounded_corners_dim }
5453 }

```

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).

```
5454 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5455 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5456 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5457 {
```

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.

```
5458 \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`.

```
5459 \str_if_in:nnF { #1 } { !! }
5460 {
5461 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5462 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5463 }
5464 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5465 {
5466 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5467 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5468 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5469 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5470 }
```

The following command must be used within a `\pgfpicture`.

```
5471 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5472 {
5473 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5474 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5475 \group_begin:
5476 \pgfsetcornersarced
5477 {
5478 \pgfpoint
5479 { \l_@@_tab_rounded_corners_dim }
5480 { \l_@@_tab_rounded_corners_dim }
5481 }
```

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.

```
5482 \bool_if:NTF \l_@@_hvlines_bool
5483 {
5484 \pgfpathrectanglecorners
```

```

5485         {
5486             \pgfpointadd
5487             { \@@_qpoint:n { row-1 } }
5488             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5489         }
5490         {
5491             \pgfpointadd
5492             {
5493                 \@@_qpoint:n
5494                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5495             }
5496             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5497         }
5498     }
5499     {
5500         \pgfpathrectanglecorners
5501         { \@@_qpoint:n { row-1 } }
5502         {
5503             \pgfpointadd
5504             {
5505                 \@@_qpoint:n
5506                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5507             }
5508             { \pgfpoint \c_zero_dim \arrayrulewidth }
5509         }
5510     }
5511     \pgfusepath { clip }
5512     \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5513     }
5514 }

```

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).

```

5515 \cs_new_protected:Npn \@@_actually_color:
5516 {
5517     \pgfpicture
5518     \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.

```

5519     \@@_clip_with_rounded_corners:
5520     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5521     {
5522         \int_compare:nNnTF { ##1 } = \c_one_int
5523         {
5524             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5525             \use:c { g_@@_color _ 1 _tl }
5526             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5527         }
5528         {
5529             \begin { pgfscope }
5530                 \@@_color_opacity ##2
5531                 \use:c { g_@@_color _ ##1 _tl }
5532                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5533                 \pgfusepath { fill }
5534             \end { pgfscope }
5535         }
5536     }
5537     \endpgfpicture
5538 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5539 \cs_new_protected:Npn \@@_color_opacity
5540 {
5541   \peek_meaning:NTF [
5542     { \@@_color_opacity:w }
5543     { \@@_color_opacity:w [ ] }
5544   }

```

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.

```

5545 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5546 {
5547   \tl_clear:N \l_tmpa_tl
5548   \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.

```

5549   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5550   \tl_if_empty:NTF \l_tmpb_tl
5551     { \@declaredcolor }
5552     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5553 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5554 \keys_define:nn { nicematrix / color-opacity }
5555 {
5556   opacity .tl_set:N          = \l_tmpa_tl ,
5557   opacity .value_required:n = true
5558 }

5559 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5560 {
5561   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5562   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5563   \@@_cartesian_path:
5564 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5565 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5566 {
5567   \tl_if_blank:nF { #2 }
5568   {
5569     \@@_add_to_colors_seq:en
5570     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5571     { \@@_cartesian_color:nn { #3 } { - } }
5572   }
5573 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5574 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5575 {
5576   \tl_if_blank:nF { #2 }
5577   {
5578     \@@_add_to_colors_seq:en
5579     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5580     { \@@_cartesian_color:nn { - } { #3 } }
5581   }
5582 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5583 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5584 {
5585   \tl_if_blank:nF { #2 }
5586   {
5587     \@@_add_to_colors_seq:en
5588     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5589     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5590   }
5591 }

```

The last argument is the radius of the corners of the rectangle.

```

5592 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5593 {
5594   \tl_if_blank:nF { #2 }
5595   {
5596     \@@_add_to_colors_seq:en
5597     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5598     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5599   }
5600 }

```

The last argument is the radius of the corners of the rectangle.

```

5601 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5602 {
5603   \@@_cut_on_hyphen:w #1 \q_stop
5604   \tl_clear_new:N \l_@@_tmpc_tl
5605   \tl_clear_new:N \l_@@_tmpd_tl
5606   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5607   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5608   \@@_cut_on_hyphen:w #2 \q_stop
5609   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5610   \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`.

```

5611 \@@_cartesian_path:n { #3 }
5612 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5613 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5614 {
5615   \clist_map_inline:nn { #3 }
5616   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5617 }

```

```

5618 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5619 {
5620   \int_step_inline:nn \c@iRow
5621   {
5622     \int_step_inline:nn \c@jCol
5623     {
5624       \int_if_even:nTF { #####1 + ##1 }
5625       { \@@_cellcolor [ #1 ] { #2 } }
5626       { \@@_cellcolor [ #1 ] { #3 } }
5627       { ##1 - #####1 }
5628     }
5629   }
5630 }

```

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”.

```

5631 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5632 {
5633   \@@_rectanglecolor [ #1 ] { #2 }
5634   { 1 - 1 }
5635   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5636 }

5637 \keys_define:nn { nicematrix / rowcolors }
5638 {
5639   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5640   respect-blocks .default:n = true ,
5641   cols .tl_set:N = \l_@@_cols_tl ,
5642   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5643   restart .default:n = true ,
5644   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5645 }

```

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*.

```

5646 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5647 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5648   \group_begin:
5649   \seq_clear_new:N \l_@@_colors_seq
5650   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5651   \tl_clear_new:N \l_@@_cols_tl
5652   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5653   \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).

```

5654   \int_zero_new:N \l_@@_color_int
5655   \int_set_eq:NN \l_@@_color_int \c_one_int
5656   \bool_if:NT \l_@@_respect_blocks_bool
5657   {

```

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`).

```

5658       \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5659       \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5660       { \@@_not_in_exterior_p:nnnnn ##1 }
5661   }
5662   \pgfpicture
5663   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5664   \clist_map_inline:nn { #2 }
5665   {
5666     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5667     \tl_if_in:NnTF \l_tmpa_tl { - }
5668     { \@@_cut_on_hyphen:w ##1 \q_stop }
5669     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, `l_tmpa_tl` and `l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

5670      \int_set:Nn \l_tmpa_int \l_tmpa_tl
5671      \int_set:Nn \l_@@_color_int
5672      { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5673      \int_zero_new:N \l_@@_tmpc_int
5674      \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5675      \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5676      {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5677      \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).

```

5678      \bool_if:NT \l_@@_respect_blocks_bool
5679      {
5680        \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5681        { \@@_intersect_our_row_p:nnnnn ###1 }
5682        \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5683      }
5684      \tl_set:No \l_@@_rows_tl
5685      { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5686      \tl_clear_new:N \l_@@_color_tl
5687      \tl_set:Ne \l_@@_color_tl
5688      {
5689        \@@_color_index:n
5690        {
5691          \int_mod:nn
5692          { \l_@@_color_int - 1 }
5693          { \seq_count:N \l_@@_colors_seq }
5694          + 1
5695        }
5696      }
5697      \tl_if_empty:NF \l_@@_color_tl
5698      {
5699        \@@_add_to_colors_seq:ee
5700        { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5701        { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5702      }
5703      \int_incr:N \l_@@_color_int
5704      \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5705    }
5706  }
5707  \endpgfpicture
5708  \group_end:
5709 }

```

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.

```

5710 \cs_new:Npn \@@_color_index:n #1
5711 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5712   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5713   { \@@_color_index:n { #1 - 1 } }
5714   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5715 }

```


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.

```
5716 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5717 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5718 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5719 {
5720   \int_compare:nNt { #3 } > \l_tmpb_int
5721   { \int_set:Nn \l_tmpb_int { #3 } }
5722 }
```

```
5723 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5724 {
5725   \int_if_zero:nTF { #4 }
5726   \prg_return_false:
5727   {
5728     \int_compare:nNtTF { #2 } > \c@jCol
5729     \prg_return_false:
5730     \prg_return_true:
5731   }
5732 }
```

The following command returns true when the block intersects the row `\l_tmpa_int`.

```
5733 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5734 {
5735   \int_compare:nNtTF { #1 } > \l_tmpa_int
5736   \prg_return_false:
5737   {
5738     \int_compare:nNtTF \l_tmpa_int > { #3 }
5739     \prg_return_false:
5740     \prg_return_true:
5741   }
5742 }
```

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`).

```
5743 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5744 {
5745   \dim_compare:nNtTF { #1 } = \c_zero_dim
5746   {
5747     \bool_if:NTF
5748     \l_@@_nocolor_used_bool
5749     \@@_cartesian_path_normal_ii:
5750     {
5751       \clist_if_empty:NTF \l_@@_corners_cells_clist
5752       { \@@_cartesian_path_normal_i:n { #1 } }
5753       \@@_cartesian_path_normal_ii:
5754     }
5755   }
5756   { \@@_cartesian_path_normal_i:n { #1 } }
5757 }
```

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.

```

5758 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5759 {
5760   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5761 \clist_map_inline:Nn \l_@@_cols_tl
5762 {
5763   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5764   \tl_if_in:NnTF \l_tmpa_tl { - }
5765     { \@@_cut_on_hyphen:w ##1 \q_stop }
5766     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5767   \tl_if_empty:NTF \l_tmpa_tl
5768     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5769     {
5770       \str_if_eq:eeT \l_tmpa_tl { * }
5771       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5772     }
5773   \int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
5774     { \@@_error:n { Invalid~col~number } }
5775   \tl_if_empty:NTF \l_tmpb_tl
5776     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5777     {
5778       \str_if_eq:eeT \l_tmpb_tl { * }
5779       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5780     }
5781   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5782     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5783   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5784   \@@_qpoint:n { col - \l_tmpa_tl }
5785   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5786     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5787     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5788   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5789   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5790 \clist_map_inline:Nn \l_@@_rows_tl
5791 {
5792   \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5793   \tl_if_in:NnTF \l_tmpa_tl { - }
5794     { \@@_cut_on_hyphen:w ####1 \q_stop }
5795     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5796   \tl_if_empty:NTF \l_tmpa_tl
5797     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5798     {
5799       \str_if_eq:eeT \l_tmpa_tl { * }
5800       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5801     }
5802   \tl_if_empty:NTF \l_tmpb_tl
5803     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5804     {
5805       \str_if_eq:eeT \l_tmpb_tl { * }
5806       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5807     }
5808   \int_compare:nNnT \l_tmpa_tl > \g_@@_row_total_int
5809     { \@@_error:n { Invalid~row~number } }
5810   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5811     { \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`.

```

5812   \cs_if_exist:cF

```

```

5813         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5814         {
5815             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5816             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5817             \@@_qpoint:n { row - \l_tmpa_tl }
5818             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5819             \pgfpathrectanglecorners
5820             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5821             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5822         }
5823     }
5824 }
5825 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5826 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5827 {
5828     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5829     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5830     \clist_map_inline:Nn \l_@@_cols_tl
5831     {
5832         \@@_qpoint:n { col - ##1 }
5833         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5834         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5835         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5836         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5837         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5838     \clist_map_inline:Nn \l_@@_rows_tl
5839     {
5840         \@@_if_in_corner:nF { #####1 - ##1 }
5841         {
5842             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5843             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5844             \@@_qpoint:n { row - #####1 }
5845             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5846             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5847             {
5848                 \pgfpathrectanglecorners
5849                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5850                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5851             }
5852         }
5853     }
5854 }
5855 }

```

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`).

```

5856 \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

```

5857 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5858 {
5859     \bool_set_true:N \l_@@_nocolor_used_bool
5860     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5861     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5862 \clist_map_inline:Nn \l_@@_rows_tl
5863 {
5864   \clist_map_inline:Nn \l_@@_cols_tl
5865   { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - #####1 } { } }
5866 }
5867 }

```

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`.

```

5868 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5869 {
5870   \clist_set_eq:NN \l_tmpa_clist #1
5871   \clist_clear:N #1
5872   \clist_map_inline:Nn \l_tmpa_clist
5873   {
5874     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5875     \tl_if_in:NnTF \l_tmpa_tl { - }
5876     { \@@_cut_on_hyphen:w ##1 \q_stop }
5877     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5878     \bool_lazy_or:nnT
5879     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5880     { \tl_if_blank_p:o \l_tmpa_tl }
5881     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5882     \bool_lazy_or:nnT
5883     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5884     { \tl_if_blank_p:o \l_tmpb_tl }
5885     { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5886     \int_compare:nNnT \l_tmpb_tl > #2
5887     { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5888     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5889     { \clist_put_right:Nn #1 { #####1 } }
5890   }
5891 }

```

The following command will be linked to `\cellcolor` in the tabular.

```

5892 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5893 {
5894   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5895   {

```

We must not expand the color (`#2`) because the color may contain the token `!` which may be activated by some packages (ex.: `babel` with the option `french` on `latex` and `pdflatex`).

```

5896     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5897     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5898   }
5899   \ignorespaces
5900 }

```

The following command will be linked to `\rowcolor` in the tabular.

```

5901 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5902 {
5903   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5904   {
5905     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5906     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5907     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5908   }
5909   \ignorespaces
5910 }

```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by currying).

```
5911 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5912 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around #2 and #3 are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
5913 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5914 {
5915   \peek_remove_spaces:n
5916   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5917 }

5918 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5919 {
```

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`.

```
5920   \seq_gclear:N \g_tmpa_seq
5921   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5922   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5923   \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`.

```
5924   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5925   {
5926     { \int_use:N \c@iRow }
5927     { \exp_not:n { #1 } }
5928     { \exp_not:n { #2 } }
5929     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5930   }
5931 }
```

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`).

```
5932 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5933 {
5934   \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).

```
5935   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5936   {
5937     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5938     {
5939       \@@_rowlistcolors
5940       [ \exp_not:n { #2 } ]
5941       { #1 - \int_eval:n { \c@iRow - 1 } }
5942       { \exp_not:n { #3 } }
5943       [ \exp_not:n { #4 } ]
5944     }
```

```

5945     }
5946 }

```

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.

```

5947 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5948 {
5949   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5950     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5951   \seq_gclear:N \g_@@_rowlistcolors_seq
5952 }

5953 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5954 {
5955   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5956     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5957 }

```

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.

```

5958 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5959 {

```

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).

```

5960   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5961   {

```

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).

```

5962     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5963     {
5964       \exp_not:N \columncolor [ #1 ]
5965       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5966     }
5967   }
5968 }

5969 \hook_gput_code:nnn { begindocument } { . }
5970 {
5971   \IfPackageLoadedTF { colortbl }
5972   {
5973     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5974     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5975     \cs_new_protected:Npn \@@_revert_colortbl:
5976     {
5977       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5978       {
5979         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5980         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5981       }
5982     }
5983   }
5984   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5985 }

```

```

5986 \cs_new_protected:Npn \@@_EmptyColumn:n #1
5987 {
5988   \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
5989     { { -2 } { #1 } { 98 } { #1 } { } } % 98 and not 99 !
5990   \columncolor { nocolor } { #1 }
5991 }

```

22 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

5992 \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`.

```

5993 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5994 {
5995   \int_if_zero:nTF \l_@@_first_col_int
5996     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5997     {
5998       \int_if_zero:nTF \c@jCol
5999         {
6000           \int_compare:nNf \c@iRow = { -1 }
6001             { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6002         }
6003         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6004       }
6005 }

```

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.

```

6006 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6007 {
6008   \int_if_zero:nF \c@iRow
6009   {
6010     \int_compare:nNf \c@iRow = \l_@@_last_row_int
6011       {
6012         \int_compare:nNt \c@jCol > \c_zero_int
6013         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6014       }
6015     }
6016 }

```

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:nNt \c@iRow < \l_@@_last_row_int`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```

6017 \cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2

```

```

6018 {
6019   \IfPackageLoadedTF { tikz }
6020   {
6021     \IfPackageLoadedTF { booktabs }
6022     { #2 }
6023     { \@@_error:nn { TopRule~without~booktabs } { #1 } }
6024   }
6025   { \@@_error:nn { TopRule~without~tikz } { #1 } }
6026 }
6027 \NewExpandableDocumentCommand { \@@_TopRule } { } { }
6028 { \@@_tikz_booktabs_loaded:nn \TopRule \@@_TopRule_i: }
6029 \cs_new:Npn \@@_TopRule_i:
6030 {
6031   \noalign \bgroup
6032   \peek_meaning:NTF [
6033   { \@@_TopRule_ii: }
6034   { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
6035 }
6036 \NewDocumentCommand \@@_TopRule_ii: { o }
6037 {
6038   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6039   {
6040     \@@_hline:n
6041     {
6042       position = \int_eval:n { \c@iRow + 1 } ,
6043       tikz =
6044       {
6045         line-width = #1 ,
6046         yshift = 0.25 \arrayrulewidth ,
6047         shorten-< = - 0.5 \arrayrulewidth
6048       } ,
6049       total-width = #1
6050     }
6051   }
6052   \skip_vertical:n { \belowrulesep + #1 }
6053   \egroup
6054 }
6055 \NewExpandableDocumentCommand { \@@_BottomRule } { } { }
6056 { \@@_tikz_booktabs_loaded:nn \BottomRule \@@_BottomRule_i: }
6057 \cs_new:Npn \@@_BottomRule_i:
6058 {
6059   \noalign \bgroup
6060   \peek_meaning:NTF [
6061   { \@@_BottomRule_ii: }
6062   { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
6063 }
6064 \NewDocumentCommand \@@_BottomRule_ii: { o }
6065 {
6066   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6067   {
6068     \@@_hline:n
6069     {
6070       position = \int_eval:n { \c@iRow + 1 } ,
6071       tikz =
6072       {
6073         line-width = #1 ,
6074         yshift = 0.25 \arrayrulewidth ,
6075         shorten-< = - 0.5 \arrayrulewidth
6076       } ,
6077       total-width = #1 ,
6078     }

```



```

6079     }
6080     \skip_vertical:N \aboverulesep
6081     \@@_create_row_node_i:
6082     \skip_vertical:n { #1 }
6083     \egroup
6084   }
6085   \NewExpandableDocumentCommand { \@@_MidRule } { } {
6086     { \@@_tikz_booktabs_loaded:nn \MidRule \@@_MidRule_i: }
6087     \cs_new:Npn \@@_MidRule_i:
6088     {
6089       \noalign \bgroup
6090       \peek_meaning:NTF [
6091         { \@@_MidRule_ii: }
6092         { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
6093       ]
6094     }
6095     \NewDocumentCommand \@@_MidRule_ii: { o }
6096     {
6097       \skip_vertical:N \aboverulesep
6098       \@@_create_row_node_i:
6099       \tl_gput_right:Ne \g_@@_pre_code_after_tl
6100       {
6101         \@@_hline:n
6102         {
6103           position = \int_eval:n { \c@iRow + 1 } ,
6104           tikz =
6105           {
6106             line-width = #1 ,
6107             yshift = 0.25 \arrayrulewidth ,
6108             shorten~< = - 0.5 \arrayrulewidth
6109           } ,
6110           total-width = #1 ,
6111         }
6112       }
6113       \skip_vertical:n { \belowrulesep + #1 }
6114     }

```

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.

```

6115 \keys_define:nn { nicematrix / Rules }
6116 {
6117   position .int_set:N = \l_@@_position_int ,
6118   position .value_required:n = true ,
6119   start .int_set:N = \l_@@_start_int ,
6120   end .code:n =
6121     \bool_lazy_or:nnTF
6122     { \tl_if_empty_p:n { #1 } }
6123     { \str_if_eq_p:ee { #1 } { last } }
6124     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6125     { \int_set:Nn \l_@@_end_int { #1 } }
6126 }

```

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.

```

6127 \keys_define:nn { nicematrix / RulesBis }
6128 {
6129   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6130   multiplicity .initial:n = 1 ,
6131   dotted .bool_set:N = \l_@@_dotted_bool ,
6132   dotted .initial:n = false ,
6133   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]`.

```

6134   color .code:n =
6135     \@@_set_CT@arc@:n { #1 }
6136     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6137   color .value_required:n = true ,
6138   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6139   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.

```

6140   tikz .code:n =
6141     \IfPackageLoadedTF { tikz }
6142       { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6143       { \@@_error:n { tikz~without~tikz } } ,
6144   tikz .value_required:n = true ,
6145   total-width .dim_set:N = \l_@@_rule_width_dim ,
6146   total-width .value_required:n = true ,
6147   width .meta:n = { total-width = #1 } ,
6148   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6149 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6150 \cs_new_protected:Npn \@@_vline:n #1
6151 {

```

The group is for the options.

```

6152   \group_begin:
6153   \int_set_eq:NN \l_@@_end_int \c@iRow
6154   \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).

```

6155   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6156     \@@_vline_i:
6157   \group_end:
6158 }
6159 \cs_new_protected:Npn \@@_vline_i:
6160 {

```

`\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`.

```

6161   \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6162   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6163     \l_tmpa_tl
6164   {

```

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.

```

6165      \bool_gset_true:N \g_tmpa_bool
6166      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6167      { \@@_test_vline_in_block:nnnnn ##1 }
6168      \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6169      { \@@_test_vline_in_block:nnnnn ##1 }
6170      \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6171      { \@@_test_vline_in_stroken_block:nnnn ##1 }
6172      \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6173      \bool_if:NTF \g_tmpa_bool
6174      {
6175          \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.

```

6176          { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6177      }
6178      {
6179          \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6180          {
6181              \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6182              \@@_vline_ii:
6183              \int_zero:N \l_@@_local_start_int
6184          }
6185      }
6186  }
6187  \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6188  {
6189      \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6190      \@@_vline_ii:
6191  }
6192  }

6193  \cs_new_protected:Npn \@@_test_in_corner_v:
6194  {
6195      \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6196      {
6197          \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6198          { \bool_set_false:N \g_tmpa_bool }
6199      }
6200      {
6201          \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6202          {
6203              \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6204              { \bool_set_false:N \g_tmpa_bool }
6205              {
6206                  \@@_if_in_corner:nT
6207                  { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6208                  { \bool_set_false:N \g_tmpa_bool }
6209              }
6210          }
6211      }
6212  }

6213  \cs_new_protected:Npn \@@_vline_ii:
6214  {
6215      \tl_clear:N \l_@@_tikz_rule_tl
6216      \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl

```

```

6217 \bool_if:NTF \l_@@_dotted_bool
6218 \@@_vline_iv:
6219 {
6220 \tl_if_empty:NTF \l_@@_tikz_rule_tl
6221 \@@_vline_iii:
6222 \@@_vline_v:
6223 }
6224 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6225 \cs_new_protected:Npn \@@_vline_iii:
6226 {
6227 \pgfpicture
6228 \pgfrememberpicturepositiononpagetrue
6229 \pgf@relevantforpicturesizefalse
6230 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6231 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6232 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6233 \dim_set:Nn \l_tmpb_dim
6234 {
6235 \pgf@x
6236 - 0.5 \l_@@_rule_width_dim
6237 +
6238 ( \arrayrulewidth * \l_@@_multiplicity_int
6239 + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6240 }
6241 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6242 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6243 \bool_lazy_all:nT
6244 {
6245 { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6246 { \cs_if_exist_p:N \CT@drsc@ }
6247 { ! \tl_if_blank_p:o \CT@drsc@ }
6248 }
6249 {
6250 \group_begin:
6251 \CT@drsc@
6252 \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6253 \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6254 \dim_set:Nn \l_@@_tmpd_dim
6255 {
6256 \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6257 * ( \l_@@_multiplicity_int - 1 )
6258 }
6259 \pgfpathrectanglecorners
6260 { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6261 { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6262 \pgfusepath { fill }
6263 \group_end:
6264 }
6265 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6266 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6267 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6268 {
6269 \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6270 \dim_sub:Nn \l_tmpb_dim \doublerulesep
6271 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6272 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6273 }
6274 \CT@arc@
6275 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6276 \pgfsetrectcap
6277 \pgfusepathqstroke

```

```

6278 \endpgfpicture
6279 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6280 \cs_new_protected:Npn \@@_vline_iv:
6281 {
6282   \pgfpicture
6283   \pgfrememberpicturepositiononpagetrue
6284   \pgf@relevantforpicturesizefalse
6285   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6286   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6287   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6288   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6289   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6290   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6291   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6292   \CT@arc@
6293   \@@_draw_line:
6294   \endpgfpicture
6295 }

```

The following code is for the case when the user uses the key `tikz`.

```

6296 \cs_new_protected:Npn \@@_vline_v:
6297 {
6298   \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).

```

6299   \CT@arc@
6300   \tl_if_empty:NF \l_@@_rule_color_tl
6301   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6302   \pgfrememberpicturepositiononpagetrue
6303   \pgf@relevantforpicturesizefalse
6304   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6305   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6306   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6307   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6308   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6309   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6310   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6311   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6312   ( \l_tmpb_dim , \l_tmpa_dim ) --
6313   ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6314   \end {tikzpicture }
6315 }

```

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).

```

6316 \cs_new_protected:Npn \@@_draw_vlines:
6317 {
6318   \int_step_inline:nnn
6319   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6320   {
6321     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6322     \c@jCol
6323     { \int_eval:n { \c@jCol + 1 } }
6324   }
6325   {
6326     \str_if_eq:eeF \l_@@_vlines_clist { all }
6327     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6328     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6329     }
6330 }

```

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}`.

```

6331 \cs_new_protected:Npn \@@_hline:n #1
6332 {

```

The group is for the options.

```

6333   \group_begin:
6334   \int_zero_new:N \l_@@_end_int
6335   \int_set_eq:NN \l_@@_end_int \c_jCol
6336   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6337   \@@_hline_i:
6338   \group_end:
6339 }
6340 \cs_new_protected:Npn \@@_hline_i:
6341 {
6342   \int_zero_new:N \l_@@_local_start_int
6343   \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`.

```

6344   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6345   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6346     \l_tmpb_tl
6347   {

```

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.

```

6348     \bool_gset_true:N \g_tmpa_bool

```

We test whether we are in a block.

```

6349     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6350       { \@@_test_hline_in_block:nnnnn ##1 }
6351     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6352       { \@@_test_hline_in_block:nnnnn ##1 }
6353     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6354       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6355     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6356     \bool_if:NTF \g_tmpa_bool
6357       {
6358         \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.

```

6359         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6360       }
6361     {
6362       \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6363       {
6364         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6365         \@@_hline_ii:
6366         \int_zero:N \l_@@_local_start_int
6367       }
6368     }
6369   }
6370   \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```

```

6371     {
6372         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6373         \@@_hline_ii:
6374     }
6375 }

6376 \cs_new_protected:Npn \@@_test_in_corner_h:
6377 {
6378     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6379     {
6380         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6381         { \bool_set_false:N \g_tmpa_bool }
6382     }
6383     {
6384         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6385         {
6386             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6387             { \bool_set_false:N \g_tmpa_bool }
6388             {
6389                 \@@_if_in_corner:nT
6390                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6391                 { \bool_set_false:N \g_tmpa_bool }
6392             }
6393         }
6394     }
6395 }

6396 \cs_new_protected:Npn \@@_hline_ii:
6397 {
6398     \tl_clear:N \l_@@_tikz_rule_tl
6399     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6400     \bool_if:NTF \l_@@_dotted_bool
6401     \@@_hline_iv:
6402     {
6403         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6404         \@@_hline_iii:
6405         \@@_hline_v:
6406     }
6407 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6408 \cs_new_protected:Npn \@@_hline_iii:
6409 {
6410     \pgfpicture
6411     \pgfrememberpicturepositiononpagetrue
6412     \pgf@relevantforpicturesizefalse
6413     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6414     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6415     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6416     \dim_set:Nn \l_tmpb_dim
6417     {
6418         \pgf@y
6419         - 0.5 \l_@@_rule_width_dim
6420         +
6421         ( \arrayrulewidth * \l_@@_multiplicity_int
6422           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6423     }
6424     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6425     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6426     \bool_lazy_all:nT
6427     {

```

```

6428     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6429     { \cs_if_exist_p:N \CT@drsc@ }
6430     { ! \tl_if_blank_p:o \CT@drsc@ }
6431   }
6432   {
6433     \group_begin:
6434     \CT@drsc@
6435     \dim_set:Nn \l_@@_tmpd_dim
6436     {
6437       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6438       * ( \l_@@_multiplicity_int - 1 )
6439     }
6440     \pgfpathrectanglecorners
6441     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6442     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6443     \pgfusepathqfill
6444     \group_end:
6445   }
6446   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6447   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6448   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6449   {
6450     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6451     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6452     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6453     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6454   }
6455   \CT@arc@
6456   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6457   \pgfsetrectcap
6458   \pgfusepathqstroke
6459   \endpgfpicture
6460 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6461 \cs_new_protected:Npn \@@_hline_iv:
6462 {
6463   \pgfpicture
6464   \pgfrememberpicturepositiononpagetrue
6465   \pgf@relevantforpicturesizefalse
6466   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6467   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6468   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6469   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6470   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x

```



```

6471 \int_compare:nNtT \l_@@_local_start_int = \c_one_int
6472 {
6473   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6474   \bool_if:NF \g_@@_delims_bool
6475   { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by $0.5 \l_@@_xdots_inter_dim$ is *ad hoc* for a better result.

```

6476   \tl_if_eq:NnF \g_@@_left_delim_tl (
6477     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6478   )
6479   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6480   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6481   \int_compare:nNtT \l_@@_local_end_int = \c@jCol
6482   {
6483     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6484     \bool_if:NF \g_@@_delims_bool
6485     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6486     \tl_if_eq:NnF \g_@@_right_delim_tl )
6487     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6488   }
6489   \CT@arc@
6490   \@@_draw_line:
6491   \endpgfpicture
6492 }

```

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`).

```

6493 \cs_new_protected:Npn \@@_hline_v:
6494 {
6495   \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).

```

6496   \CT@arc@
6497   \tl_if_empty:NF \l_@@_rule_color_tl
6498   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6499   \pgfrememberpicturepositiononpagetrue
6500   \pgf@relevantforpicturesizefalse
6501   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6502   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6503   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6504   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6505   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6506   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6507   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6508   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6509   ( \l_tmpa_dim , \l_tmpb_dim ) --
6510   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6511   \end { tikzpicture }
6512 }

```

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).

```

6513 \cs_new_protected:Npn \@@_draw_hlines:
6514 {
6515   \int_step_inline:nnn
6516   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6517   {
6518     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool

```

```

6519         \c@iRow
6520         { \int_eval:n { \c@iRow + 1 } }
6521     }
6522     {
6523         \str_if_eq:eeF \l_@@_hlines_clist { all }
6524         { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6525         { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6526     }
6527 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6528 \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.

```

6529 \cs_set:Npn \@@_Hline_i:n #1
6530 {
6531     \peek_remove_spaces:n
6532     {
6533         \peek_meaning:NTF \Hline
6534         { \@@_Hline_ii:nn { #1 + 1 } }
6535         { \@@_Hline_iii:n { #1 } }
6536     }
6537 }
6538 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6539 \cs_set:Npn \@@_Hline_iii:n #1
6540 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6541 \cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
6542 {
6543     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6544     \skip_vertical:N \l_@@_rule_width_dim
6545     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6546     {
6547         \@@_hline:n
6548         {
6549             multiplicity = #1 ,
6550             position = \int_eval:n { \c@iRow + 1 } ,
6551             total-width = \dim_use:N \l_@@_rule_width_dim ,
6552             #2
6553         }
6554     }
6555     \egroup
6556 }

```

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`).

```

6557 \cs_new_protected:Npn \@@_custom_line:n #1
6558 {
6559     \str_clear_new:N \l_@@_command_str
6560     \str_clear_new:N \l_@@_ccommand_str
6561     \str_clear_new:N \l_@@_letter_str
6562     \tl_clear_new:N \l_@@_other_keys_tl
6563     \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.

```

6564 \bool_lazy_all:nTF
6565 {
6566   { \str_if_empty_p:N \l_@@_letter_str }
6567   { \str_if_empty_p:N \l_@@_command_str }
6568   { \str_if_empty_p:N \l_@@_ccommand_str }
6569 }
6570 { \@@_error:n { No~letter~and~no~command } }
6571 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6572 }
6573 \keys_define:nn { nicematrix / custom-line }
6574 {
6575   letter .str_set:N = \l_@@_letter_str ,
6576   letter .value_required:n = true ,
6577   command .str_set:N = \l_@@_command_str ,
6578   command .value_required:n = true ,
6579   ccommand .str_set:N = \l_@@_ccommand_str ,
6580   ccommand .value_required:n = true ,
6581 }
6582 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6583 \cs_new_protected:Npn \@@_custom_line_i:n #1
6584 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6585 \bool_set_false:N \l_@@_tikz_rule_bool
6586 \bool_set_false:N \l_@@_dotted_rule_bool
6587 \bool_set_false:N \l_@@_color_bool
6588 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6589 \bool_if:NT \l_@@_tikz_rule_bool
6590 {
6591   \IfPackageLoadedF { tikz }
6592   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6593   \bool_if:NT \l_@@_color_bool
6594   { \@@_error:n { color~in~custom~line~with~tikz } }
6595 }
6596 \bool_if:NT \l_@@_dotted_rule_bool
6597 {
6598   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6599   { \@@_error:n { key~multiplicity~with~dotted } }
6600 }
6601 \str_if_empty:NF \l_@@_letter_str
6602 {
6603   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6604   { \@@_error:n { Several~letters } }
6605   {
6606     \tl_if_in:NoTF
6607     \c_@@_forbidden_letters_str
6608     \l_@@_letter_str
6609     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6610   }

```

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.

```

6611 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6612 { \@@_v_custom_line:n { #1 } }
6613 }
6614 }
6615 }
6616 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6617 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6618 }

```

```

6619 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6620 \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.

```

6621 \keys_define:nn { nicematrix / custom-line-bis }
6622 {
6623   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6624   multiplicity .initial:n = 1 ,
6625   multiplicity .value_required:n = true ,
6626   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6627   color .value_required:n = true ,
6628   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6629   tikz .value_required:n = true ,
6630   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6631   dotted .value_forbidden:n = true ,
6632   total-width .code:n = { } ,
6633   total-width .value_required:n = true ,
6634   width .code:n = { } ,
6635   width .value_required:n = true ,
6636   sep-color .code:n = { } ,
6637   sep-color .value_required:n = true ,
6638   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6639 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6640 \bool_new:N \l_@@_dotted_rule_bool
6641 \bool_new:N \l_@@_tikz_rule_bool
6642 \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`.

```

6643 \keys_define:nn { nicematrix / custom-line-width }
6644 {
6645   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6646   multiplicity .initial:n = 1 ,
6647   multiplicity .value_required:n = true ,
6648   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6649   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6650   \bool_set_true:N \l_@@_total_width_bool ,
6651   total-width .value_required:n = true ,
6652   width .meta:n = { total-width = #1 } ,
6653   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6654 }

```

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`).

```

6655 \cs_new_protected:Npn \@@_h_custom_line:n #1
6656 {

```

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`).

```

6657   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6658   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6659 }

```

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`).

```
6660 \cs_new_protected:Npn \@@_c_custom_line:n #1
6661 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6662   \exp_args:Nc \NewExpandableDocumentCommand
6663     { nicematrix - \l_@@_ccommand_str }
6664     { 0 { } m }
6665     {
6666       \noalign
6667       {
6668         \@@_compute_rule_width:n { #1 , ##1 }
6669         \skip_vertical:n { \l_@@_rule_width_dim }
6670         \clist_map_inline:nn
6671           { ##2 }
6672           { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6673       }
6674     }
6675   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6676 }
```

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*.

```
6677 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6678 {
6679   \tl_if_in:nnTF { #2 } { - }
6680     { \@@_cut_on_hyphen:w #2 \q_stop }
6681     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6682   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6683     {
6684       \@@_hline:n
6685       {
6686         #1 ,
6687         start = \l_tmpa_tl ,
6688         end = \l_tmpb_tl ,
6689         position = \int_eval:n { \c@iRow + 1 } ,
6690         total-width = \dim_use:N \l_@@_rule_width_dim
6691       }
6692     }
6693 }
6694 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6695 {
6696   \bool_set_false:N \l_@@_tikz_rule_bool
6697   \bool_set_false:N \l_@@_total_width_bool
6698   \bool_set_false:N \l_@@_dotted_rule_bool
6699   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6700   \bool_if:NF \l_@@_total_width_bool
6701   {
6702     \bool_if:NTF \l_@@_dotted_rule_bool
6703       { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6704       {
6705         \bool_if:NF \l_@@_tikz_rule_bool
6706         {
6707           \dim_set:Nn \l_@@_rule_width_dim
6708             {
6709               \arrayrulewidth * \l_@@_multiplicity_int
6710               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6711             }
6712         }
6713       }
6714   }
6715 }
```

```

6716 \cs_new_protected:Npn \@@_v_custom_line:n #1
6717 {
6718   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6719   \tl_gput_right:Ne \g_@@_array_preamble_tl
6720     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6721   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6722     {
6723     \@@_vline:n
6724     {
6725       #1 ,
6726       position = \int_eval:n { \c@jCol + 1 } ,
6727       total-width = \dim_use:N \l_@@_rule_width_dim
6728     }
6729   }
6730   \@@_rec_preamble:n
6731 }
6732 \@@_custom_line:n
6733 { 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.

```

6734 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6735 {
6736   \int_compare:nNnT \l_tmpa_tl > { #1 }
6737   {
6738     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6739     {
6740       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6741       {
6742         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6743         { \bool_gset_false:N \g_tmpa_bool }
6744       }
6745     }
6746   }
6747 }

```

The same for vertical rules.

```

6748 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6749 {
6750   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6751   {
6752     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6753     {
6754       \int_compare:nNnT \l_tmpb_tl > { #2 }
6755       {
6756         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6757         { \bool_gset_false:N \g_tmpa_bool }
6758       }
6759     }
6760   }
6761 }
6762 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6763 {
6764   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6765   {
6766     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6767     {

```

```

6768         \int_compare:nNnTF \l_tmpa_tl = { #1 }
6769         { \bool_gset_false:N \g_tmpa_bool }
6770         {
6771             \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6772             { \bool_gset_false:N \g_tmpa_bool }
6773         }
6774     }
6775 }
6776 }
6777 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6778 {
6779     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6780     {
6781         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6782         {
6783             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6784             { \bool_gset_false:N \g_tmpa_bool }
6785             {
6786                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6787                 { \bool_gset_false:N \g_tmpa_bool }
6788             }
6789         }
6790     }
6791 }

```

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.

```

6792 \cs_new_protected:Npn \@@_compute_corners:
6793 {
6794     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6795     { \@@_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`).

```

6796     \clist_clear:N \l_@@_corners_cells_clist
6797     \clist_map_inline:Nn \l_@@_corners_clist
6798     {
6799         \str_case:nnF { ##1 }
6800         {
6801             { NW }
6802             { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6803             { NE }
6804             { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6805             { SW }
6806             { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6807             { SE }
6808             { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6809         }
6810         { \@@_error:nn { bad~corner } { ##1 } }
6811     }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6812     \clist_if_empty:NF \l_@@_corners_cells_clist
6813     {

```

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.

```

6814     \tl_gput_right:Ne \g_@@_aux_tl
6815     {
6816         \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6817         { \l_@@_corners_cells_clist }
6818     }
6819 }
6820 }

6821 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6822 {
6823     \int_step_inline:nnn { #1 } { #3 }
6824     {
6825         \int_step_inline:nnn { #2 } { #4 }
6826         { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
6827     }
6828 }

6829 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6830 {
6831     \cs_if_exist:cTF
6832     { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
6833     \prg_return_true:
6834     \prg_return_false:
6835 }

```

“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.

```

6836 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6837 {

```

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.

```

6838     \bool_set_false:N \l_tmpa_bool
6839     \int_zero_new:N \l_@@_last_empty_row_int
6840     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6841     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6842     {
6843         \bool_lazy_or:nnTF
6844         {
6845             \cs_if_exist_p:c
6846             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6847         }
6848         { \@@_if_in_block_p:nn { ##1 } { #2 } }
6849         { \bool_set_true:N \l_tmpa_bool }
6850     }

```



```

6851         \bool_if:NF \l_tmpa_bool
6852         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6853     }
6854 }

```

Now, you determine the last empty cell in the row of number 1.

```

6855     \bool_set_false:N \l_tmpa_bool
6856     \int_zero_new:N \l_@@_last_empty_column_int
6857     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6858     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6859     {
6860         \bool_lazy_or:nnTF
6861         {
6862             \cs_if_exist_p:c
6863             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6864         }
6865         { \@@_if_in_block_p:nn { #1 } { ##1 } }
6866         { \bool_set_true:N \l_tmpa_bool }
6867         {
6868             \bool_if:NF \l_tmpa_bool
6869             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6870         }
6871     }

```

Now, we loop over the rows.

```

6872     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6873     {

```

We treat the row number ##1 with another loop.

```

6874         \bool_set_false:N \l_tmpa_bool
6875         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6876         {
6877             \bool_lazy_or:nnTF
6878             { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6879             { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6880             { \bool_set_true:N \l_tmpa_bool }
6881             {
6882                 \bool_if:NF \l_tmpa_bool
6883                 {
6884                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6885                     \clist_put_right:Nn
6886                     \l_@@_corners_cells_clist
6887                     { ##1 - #####1 }
6888                     \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6889                 }
6890             }
6891         }
6892     }
6893 }

```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```

6894 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6895 \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.

```
6896 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
6897 \keys_define:nn { nicematrix / NiceMatrixBlock }
6898 {
6899   auto-columns-width .code:n =
6900   {
6901     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6902     \dim_gzero_new:N \g_@@_max_cell_width_dim
6903     \bool_set_true:N \l_@@_auto_columns_width_bool
6904   }
6905 }

6906 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6907 {
6908   \int_gincr:N \g_@@_NiceMatrixBlock_int
6909   \dim_zero:N \l_@@_columns_width_dim
6910   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6911   \bool_if:NT \l_@@_block_auto_columns_width_bool
6912   {
6913     \cs_if_exist:cT
6914     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6915     {
6916       \dim_set:Nn \l_@@_columns_width_dim
6917       {
6918         \use:c
6919         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6920       }
6921     }
6922   }
6923 }
```

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`).

```
6924 {
6925   \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.

```
6926   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6927   {
6928     \bool_if:NT \l_@@_block_auto_columns_width_bool
6929     {
6930       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6931       \iow_shipout:Ne \@mainaux
6932       {
6933         \cs_gset:cpn
6934         { @@ _ 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.

```
6935         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6936       }
6937       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6938     }
6939   }
6940   \ignorespacesafterend
6941 }
```

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).

```

6942 \cs_new_protected:Npn \@@_create_extra_nodes:
6943 {
6944   \bool_if:nTF \l_@@_medium_nodes_bool
6945   {
6946     \bool_if:NTF \l_@@_no_cell_nodes_bool
6947     { \@@_error:n { extra-nodes~with~no-cell-nodes } }
6948     {
6949       \bool_if:NTF \l_@@_large_nodes_bool
6950       \@@_create_medium_and_large_nodes:
6951       \@@_create_medium_nodes:
6952     }
6953   }
6954   {
6955     \bool_if:NT \l_@@_large_nodes_bool
6956     {
6957       \bool_if:NTF \l_@@_no_cell_nodes_bool
6958       { \@@_error:n { extra-nodes~with~no-cell-nodes } }
6959       \@@_create_large_nodes:
6960     }
6961   }
6962 }

```

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`.

```

6963 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6964 {
6965   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6966   {
6967     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6968     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6969     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6970     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6971   }
6972   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6973   {
6974     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6975     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6976     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6977     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6978   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6979 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6980 {
6981   \int_step_variable:nnNn
6982   \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.

```

6983 {
6984   \cs_if_exist:cT
6985   { 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`.

```

6986 {
6987   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6988   \dim_set:cn { l_@@_row _ \@@_i: _ min_dim }
6989   { \dim_min:vn { l_@@_row _ \@@_i: _ min_dim } \pgf@y }
6990   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6991   {
6992     \dim_set:cn { l_@@_column _ \@@_j: _ min_dim }
6993     { \dim_min:vn { l_@@_column _ \@@_j: _ min_dim } \pgf@x }
6994   }

```

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`.

```

6995   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6996   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6997   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6998   \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6999   {
7000     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
7001     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
7002   }
7003 }
7004 }
7005 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

7006 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7007 {
7008   \dim_compare:nNnT
7009   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
7010   {
7011     \@@_qpoint:n { row - \@@_i: - base }
7012     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
7013     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
7014   }
7015 }
7016 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7017 {
7018   \dim_compare:nNnT
7019   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
7020   {
7021     \@@_qpoint:n { col - \@@_j: }
7022     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
7023     \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
7024   }
7025 }
7026 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

7027 \cs_new_protected:Npn \@@_create_medium_nodes:
7028 {
7029   \pgfpicture
7030     \pgfrememberpicturerepositiononpagetrue
7031     \pgf@relevantforpicturesizefalse
7032     \@@_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”.

```

7033     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7034     \@@_create_nodes:
7035     \endpgfpicture
7036   }

```

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:`.

```

7037 \cs_new_protected:Npn \@@_create_large_nodes:
7038 {
7039   \pgfpicture
7040     \pgfrememberpicturerepositiononpagetrue
7041     \pgf@relevantforpicturesizefalse
7042     \@@_computations_for_medium_nodes:
7043     \@@_computations_for_large_nodes:
7044     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
7045     \@@_create_nodes:
7046   \endpgfpicture
7047 }

7048 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
7049 {
7050   \pgfpicture
7051     \pgfrememberpicturerepositiononpagetrue
7052     \pgf@relevantforpicturesizefalse
7053     \@@_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”.

```

7054     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
7055     \@@_create_nodes:
7056     \@@_computations_for_large_nodes:
7057     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
7058     \@@_create_nodes:
7059   \endpgfpicture
7060 }

```

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

7061 \cs_new_protected:Npn \@@_computations_for_large_nodes:
7062 {
7063   \int_set_eq:NN \l_@@_first_row_int \c_one_int
7064   \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`.

```

7065   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7066   {
7067     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7068     {
7069     (
7070         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7071         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7072     )
7073     / 2
7074     }
7075     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7076     { l_@@_row\_@@_i: _ min_dim }
7077 }
7078 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7079 {
7080     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7081     {
7082     (
7083         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7084         \dim_use:c
7085         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7086     )
7087     / 2
7088     }
7089     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7090     { l_@@_column _ \@@_j: _ max _ dim }
7091 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7092     \dim_sub:cn
7093     { l_@@_column _ 1 _ min _ dim }
7094     \l_@@_left_margin_dim
7095     \dim_add:cn
7096     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7097     \l_@@_right_margin_dim
7098 }

```

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).

```

7099 \cs_new_protected:Npn \@@_create_nodes:
7100 {
7101     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7102     {
7103         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7104         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7105         \@@_pgf_rect_node:nnnnn
7106         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7107         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7108         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7109         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7110         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7111         \str_if_empty:NF \l_@@_name_str
7112         {
7113             \pgfnodealias
7114             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7115             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7116         }
7117     }
7118 }

```

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 .

```

7119     \seq_map_pairwise_function:NNN
7120     \g_@@_multicolumn_cells_seq
7121     \g_@@_multicolumn_sizes_seq
7122     \@@_node_for_multicolumn:nn
7123 }

7124 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7125 {
7126   \cs_set_nopar:Npn \@@_i: { #1 }
7127   \cs_set_nopar:Npn \@@_j: { #2 }
7128 }
```

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”).

```

7129 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7130 {
7131   \@@_extract_coords_values: #1 \q_stop
7132   \@@_pgf_rect_node:nnnnn
7133   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7134   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7135   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7136   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7137   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7138   \str_if_empty:NF \l_@@_name_str
7139   {
7140     \pgfnodealias
7141     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7142     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7143   }
7144 }
```

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).

```

7145 \keys_define:nn { nicematrix / Block / FirstPass }
7146 {
7147   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7148           \bool_set_true:N \l_@@_p_block_bool ,
7149   j .value_forbidden:n = true ,
7150   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7151   l .value_forbidden:n = true ,
7152   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7153   r .value_forbidden:n = true ,
7154   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7155   c .value_forbidden:n = true ,
7156   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7157   L .value_forbidden:n = true ,
7158   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,

```

```

7159 R.value_forbidden:n = true ,
7160 C.code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7161 C.value_forbidden:n = true ,
7162 t.code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7163 t.value_forbidden:n = true ,
7164 T.code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7165 T.value_forbidden:n = true ,
7166 b.code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7167 b.value_forbidden:n = true ,
7168 B.code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7169 B.value_forbidden:n = true ,
7170 m.code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7171 m.value_forbidden:n = true ,
7172 v-center .meta:n = m ,
7173 p.code:n = \bool_set_true:N \l_@@_p_block_bool ,
7174 p.value_forbidden:n = true ,
7175 color .code:n =
7176   \@@_color:n { #1 }
7177   \tl_set_rescan:Nnn
7178     \l_@@_draw_tl
7179     { \char_set_catcode_other:N ! }
7180     { #1 } ,
7181 color .value_required:n = true ,
7182 respect-arraystretch .code:n =
7183   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7184 respect-arraystretch .value_forbidden:n = true ,
7185 }

```

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.

```

7186 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7187 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7188 {

```

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).

```

7189   \peek_remove_spaces:n
7190   {
7191     \tl_if_blank:nTF { #2 }
7192     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7193     {
7194       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7195       \@@_Block_i_czech \@@_Block_i
7196       #2 \q_stop
7197     }
7198     { #1 } { #3 } { #4 }
7199   }
7200 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7201 \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`.

```

7202 {
7203   \char_set_catcode_active:N -
7204   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7205 }

```


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.

```
7206 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7207 {
```

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).

```
7208 \bool_lazy_or:nnTF
7209 { \tl_if_blank_p:n { #1 } }
7210 { \str_if_eq_p:ee { * } { #1 } }
7211 { \int_set:Nn \l_tmpa_int { 100 } }
7212 { \int_set:Nn \l_tmpa_int { #1 } }
7213 \bool_lazy_or:nnTF
7214 { \tl_if_blank_p:n { #2 } }
7215 { \str_if_eq_p:ee { * } { #2 } }
7216 { \int_set:Nn \l_tmpb_int { 100 } }
7217 { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7218 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7219 {
7220   \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7221   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7222   { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7223 }
7224 { \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.

```
7225 \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7226 \tl_set:Ne \l_tmpa_tl
7227 {
7228   { \int_use:N \c@iRow }
7229   { \int_use:N \c@jCol }
7230   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7231   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7232 }
```

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).

```
7233 \bool_set_false:N \l_tmpa_bool
7234 \bool_if:NT \l_@@_amp_in_blocks_bool
```

\tl_if_in:nnT is slightly faster than \str_if_in:nnT.

```
7235 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7236 \bool_case:nF
7237 {
7238   \l_tmpa_bool { \@@_Block_vii:eennn }
7239   \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.

```

7240      \l_@@_X_bool                                { \@@_Block_v:eennn }
7241      { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7242      { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7243      { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7244    }
7245    { \@@_Block_v:eennn }
7246    { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7247  }

```

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.

```

7248 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7249 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7250 {
7251   \int_gincr:N \g_@@_block_box_int
7252   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7253   {
7254     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7255     {
7256       \@@_actually_diagbox:nnnnnn
7257       { \int_use:N \c_iRow }
7258       { \int_use:N \c_jCol }
7259       { \int_eval:n { \c_iRow + #1 - 1 } }
7260       { \int_eval:n { \c_jCol + #2 - 1 } }
7261       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7262       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7263     }
7264   }
7265   \box_gclear_new:c
7266   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the \Block in a TeX box. *Be careful:* if after the construction of the box, the boolean \g_@@_rotate_bool is raised (which means that the command \rotate was present in the content of the \Block) we will rotate the box but also, maybe, change the position of the baseline!

```

7267 \hbox_gset:cn
7268 { g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }
7269 {

```

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}).

```

7270 \tl_if_empty:NTF \l_@@_color_tl
7271 { \int_compare:nNt { #2 } = \c_one_int \set@color }
7272 { \@@_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.

```

7273 \int_compare:nNnT { #1 } = \c_one_int
7274 {
7275     \int_if_zero:nTF \c@iRow
7276     {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\\
-2 & 3 & -4 & 5 & \\\
3 & -4 & 5 & -6 & \\\
-4 & 5 & -6 & 7 & \\\
5 & -6 & 7 & -8 & \\\
\end{bNiceMatrix}$

```

```

7277 \cs_set_eq:NN \Block \@@_NullBlock:
7278 \l_@@_code_for_first_row_tl
7279 }
7280 {
7281     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7282     {
7283         \cs_set_eq:NN \Block \@@_NullBlock:
7284         \l_@@_code_for_last_row_tl
7285     }
7286 }
7287 \g_@@_row_style_tl
7288 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7289 \@@_reset_arraystretch:
7290 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7291 #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.).

```

7292 \@@_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}`.

```

7293 \bool_if:NTF \l_@@_tabular_bool
7294 {
7295     \bool_lazy_all:nTF
7296     {
7297         { \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.

```

7298         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7299         { ! \g_@@_rotate_bool }
7300     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7301     {
7302         \use:e
7303     {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7304         \exp_not:N \begin { minipage }%
7305         [ \str_lowercase:o \l_@@_vpos_block_str ]
7306         { \l_@@_col_width_dim }
7307         \str_case:on \l_@@_hpos_block_str
7308         { c \centering r \raggedleft l \raggedright }
7309     }
7310     #5
7311     \end { minipage }
7312 }

```

In the other cases, we use a `{tabular}`.

```

7313     {
7314         \bool_if:NT \c_@@_testphase_table_bool
7315         { \tagpdfsetup { table / tagging = presentation } }
7316         \use:e
7317         {
7318             \exp_not:N \begin { tabular }%
7319             [ \str_lowercase:o \l_@@_vpos_block_str ]
7320             { @ { } \l_@@_hpos_block_str @ { } }
7321         }
7322         #5
7323         \end { tabular }
7324     }
7325 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7326     {
7327         \c_math_toggle_token
7328         \use:e
7329         {
7330             \exp_not:N \begin { array }%
7331             [ \str_lowercase:o \l_@@_vpos_block_str ]
7332             { @ { } \l_@@_hpos_block_str @ { } }
7333         }
7334         #5
7335         \end { array }
7336         \c_math_toggle_token
7337     }
7338 }

```

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).

```

7339     \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.

```

7340     \int_compare:nNnT { #2 } = \c_one_int
7341     {
7342         \dim_gset:Nn \g_@@_blocks_wd_dim
7343         {

```

```

7344         \dim_max:nn
7345         \g_@@_blocks_wd_dim
7346         {
7347             \box_wd:c
7348             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7349         }
7350     }
7351 }

```

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 explicetely an option of vertical position.

```

7352     \bool_lazy_and:nnT
7353     { \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.

```

7354     { \str_if_empty_p:N \l_@@_vpos_block_str }
7355     {
7356         \dim_gset:Nn \g_@@_blocks_ht_dim
7357         {
7358             \dim_max:nn
7359             \g_@@_blocks_ht_dim
7360             {
7361                 \box_ht:c
7362                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7363             }
7364         }
7365         \dim_gset:Nn \g_@@_blocks_dp_dim
7366         {
7367             \dim_max:nn
7368             \g_@@_blocks_dp_dim
7369             {
7370                 \box_dp:c
7371                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7372             }
7373         }
7374     }
7375     \seq_gput_right:Ne \g_@@_blocks_seq
7376     {
7377         \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.

```

7378     {
7379         \exp_not:n { #3 } ,
7380         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7381         \bool_if:NT \g_@@_rotate_bool
7382         {
7383             \bool_if:NTF \g_@@_rotate_c_bool
7384             { m }
7385             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7386         }
7387     }
7388     {
7389         \box_use_drop:c
7390         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7391     }
7392 }
7393 \bool_set_false:N \g_@@_rotate_c_bool
7394 }

```

```

7395 \cs_new:Npn \@@_adjust_hpos_rotate:
7396 {
7397   \bool_if:NT \g_@@_rotate_bool
7398   {
7399     \str_set:Ne \l_@@_hpos_block_str
7400     {
7401       \bool_if:NTF \g_@@_rotate_c_bool
7402       { c }
7403       {
7404         \str_case:onF \l_@@_vpos_block_str
7405         { b l B l t r T r }
7406         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7407       }
7408     }
7409   }
7410 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7411 \cs_new_protected:Npn \@@_rotate_box_of_block:
7412 {
7413   \box_grotate:cn
7414   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7415   { 90 }
7416   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7417   {
7418     \vbox_gset_top:cn
7419     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7420     {
7421       \skip_vertical:n { 0.8 ex }
7422       \box_use:c
7423       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7424     }
7425   }
7426   \bool_if:NT \g_@@_rotate_c_bool
7427   {
7428     \hbox_gset:cn
7429     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7430     {
7431       \c_math_toggle_token
7432       \vcenter
7433       {
7434         \box_use:c
7435         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7436       }
7437       \c_math_toggle_token
7438     }
7439   }
7440 }

```

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.

```

7441 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7442 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7443 {
7444   \seq_gput_right:Ne \g_@@_blocks_seq
7445   {

```

```

7446 \l_tmpa_tl
7447 { \exp_not:n { #3 } }
7448 {
7449   \bool_if:NTF \l_@@_tabular_bool
7450   {
7451     \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7452 \@@_reset_arraystretch:
7453 \exp_not:n
7454 {
7455   \dim_zero:N \extrarowheight
7456   #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).

```

7457 \bool_if:NT \c_@@_testphase_table_bool
7458 { \tag_stop:n { table } }
7459 \use:e
7460 {
7461   \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7462   { @ { } \l_@@_hpos_block_str @ { } }
7463   }
7464   #5
7465   \end { tabular }
7466 }
7467 \group_end:
7468 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7469 {
7470   \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7471 \@@_reset_arraystretch:
7472 \exp_not:n
7473 {
7474   \dim_zero:N \extrarowheight
7475   #4
7476   \c_math_toggle_token
7477   \use:e
7478   {
7479     \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7480     { @ { } \l_@@_hpos_block_str @ { } }
7481     }
7482     #5
7483     \end { array }
7484     \c_math_toggle_token
7485   }
7486   \group_end:
7487 }
7488 }
7489 }
7490 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7491 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7492 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7493 {
7494   \seq_gput_right:Ne \g_@@_blocks_seq
7495   {

```

```

7496     \l_tmpa_tl
7497     { \exp_not:n { #3 } }

```

Here, the curly braces for the group are mandatory.

```

7498     { { \exp_not:n { #4 #5 } } }
7499   }
7500 }

```

The following macro is also for the case of a `\Block` which uses the key `p`.

```

7501 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7502 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7503 {
7504   \seq_gput_right:Ne \g_@@_blocks_seq
7505   {
7506     \l_tmpa_tl
7507     { \exp_not:n { #3 } }
7508     { \exp_not:n { #4 #5 } }
7509   }
7510 }

```

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).

```

7511 \keys_define:nn { nicematrix / Block / SecondPass }
7512 {
7513   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7514   ampersand-in-blocks .default:n = true ,
7515   &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7516   tikz .code:n =
7517     \IfPackageLoadedTF { tikz }
7518     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7519     { \@@_error:n { tikz~key~without~tikz } } ,
7520   tikz .value_required:n = true ,
7521   fill .code:n =
7522     \tl_set_rescan:Nnn
7523     \l_@@_fill_tl
7524     { \char_set_catcode_other:N ! }
7525     { #1 } ,
7526   fill .value_required:n = true ,
7527   opacity .tl_set:N = \l_@@_opacity_tl ,
7528   opacity .value_required:n = true ,
7529   draw .code:n =
7530     \tl_set_rescan:Nnn
7531     \l_@@_draw_tl
7532     { \char_set_catcode_other:N ! }
7533     { #1 } ,
7534   draw .default:n = default ,
7535   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7536   rounded-corners .default:n = 4 pt ,
7537   color .code:n =
7538     \@@_color:n { #1 }
7539     \tl_set_rescan:Nnn
7540     \l_@@_draw_tl
7541     { \char_set_catcode_other:N ! }
7542     { #1 } ,
7543   borders .clist_set:N = \l_@@_borders_clist ,
7544   borders .value_required:n = true ,
7545   hvlines .meta:n = { vlines , hlines } ,
7546   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7547   vlines .default:n = true ,

```



```

7548 hlines .bool_set:N = \l_@@_hlines_block_bool,
7549 hlines .default:n = true ,
7550 line-width .dim_set:N = \l_@@_line_width_dim ,
7551 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7552 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7553 \bool_set_true:N \l_@@_p_block_bool ,
7554 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7555 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7556 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7557 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7558 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7559 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7560 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7561 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7562 \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7563 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7564 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7565 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7566 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7567 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7568 m .value_forbidden:n = true ,
7569 v-center .meta:n = m ,
7570 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7571 p .value_forbidden:n = true ,
7572 name .tl_set:N = \l_@@_block_name_str ,
7573 name .value_required:n = true ,
7574 name .initial:n = ,
7575 respect-arraystretch .code:n =
7576 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7577 respect-arraystretch .value_forbidden:n = true ,
7578 transparent .bool_set:N = \l_@@_transparent_bool ,
7579 transparent .default:n = true ,
7580 transparent .initial:n = false ,
7581 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7582 }

```

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.

```

7583 \cs_new_protected:Npn \@@_draw_blocks:
7584 {
7585 \bool_if:NTF \c_@@_recent_array_bool
7586 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7587 { \cs_set_eq:NN \ialign \@@_old_ialign: }
7588 \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7589 }
7590 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7591 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7592 \int_zero_new:N \l_@@_last_row_int
7593 \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format *i-j*. However, the user is allowed to omit *i* or *j* (or both). This will be interpreted as follows: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now (we write 98 for the case the the command `\Block` has been issued in the “first row”).

```

7594 \int_compare:nNnTF { #3 } > { 98 }

```

```

7595     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7596     { \int_set:Nn \l_@@_last_row_int { #3 } }
7597 \int_compare:nNnTF { #4 } > { 98 }
7598     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7599     { \int_set:Nn \l_@@_last_col_int { #4 } }
7600 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7601 {
7602     \bool_lazy_and:nnTF
7603     \l_@@_preamble_bool
7604     {
7605         \int_compare_p:n
7606         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7607     }
7608     {
7609         \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7610         \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7611         \@@_msg_redirect_name:nn { columns-not-used } { none }
7612     }
7613     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7614 }
7615 {
7616     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7617     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7618     {
7619         \@@_Block_v:nneenn
7620         { #1 }
7621         { #2 }
7622         { \int_use:N \l_@@_last_row_int }
7623         { \int_use:N \l_@@_last_col_int }
7624         { #5 }
7625         { #6 }
7626     }
7627 }
7628 }

```

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

```

7629 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7630 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7631 {

```

The group is for the keys.

```

7632     \group_begin:
7633     \int_compare:nNnT { #1 } = { #3 }
7634     { \str_set:Nn \l_@@_vpos_block_str { t } }
7635     \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`.

```

7636     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7637
7638     \bool_lazy_and:nnT
7639     \l_@@_vlines_block_bool
7640     { ! \l_@@_ampersand_bool }
7641     {
7642         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7643         {
7644             \@@_vlines_block:nnn
7645             { \exp_not:n { #5 } }
7646             { #1 - #2 }
7647             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7648         }

```

```

7649 \bool_if:NT \l_@@_hlines_block_bool
7650 {
7651   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7652   {
7653     \@@_hlines_block:nnn
7654     { \exp_not:n { #5 } }
7655     { #1 - #2 }
7656     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7657   }
7658 }
7659 \bool_if:NF \l_@@_transparent_bool
7660 {
7661   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7662   {

```

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).

```

7663   \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7664   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7665   }
7666 }

7667 \tl_if_empty:NF \l_@@_draw_tl
7668 {
7669   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7670   { \@@_error:n { hlines-with-color } }
7671   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7672   {
7673     \@@_stroke_block:nnn

```

#5 are the options

```

7674     { \exp_not:n { #5 } }
7675     { #1 - #2 }
7676     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7677   }
7678   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7679   { { #1 } { #2 } { #3 } { #4 } }
7680 }

7681 \clist_if_empty:NF \l_@@_borders_clist
7682 {
7683   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7684   {
7685     \@@_stroke_borders_block:nnn
7686     { \exp_not:n { #5 } }
7687     { #1 - #2 }
7688     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7689   }
7690 }

7691 \tl_if_empty:NF \l_@@_fill_tl
7692 {
7693   \@@_add_opacity_to_fill:
7694   \tl_gput_right:Ne \g_@@_pre_code_before_tl
7695   {
7696     \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7697     { #1 - #2 }
7698     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7699     { \dim_use:N \l_@@_rounded_corners_dim }
7700   }
7701 }

7702 \seq_if_empty:NF \l_@@_tikz_seq
7703 {
7704   \tl_gput_right:Ne \g_nicematrix_code_before_tl

```

```

7705     {
7706         \@@_block_tikz:nnnnn
7707         { \seq_use:Nn \l_@@_tikz_seq { , } }
7708         { #1 }
7709         { #2 }
7710         { \int_use:N \l_@@_last_row_int }
7711         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7712     }
7713 }

7714 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7715 {
7716     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7717     {
7718         \@@_actually_diagbox:nnnnnn
7719         { #1 }
7720         { #2 }
7721         { \int_use:N \l_@@_last_row_int }
7722         { \int_use:N \l_@@_last_col_int }
7723         { \exp_not:n { ##1 } }
7724         { \exp_not:n { ##2 } }
7725     }
7726 }

```

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.

```

7727 \pgfpicture
7728 \pgfrememberpicturepositiononpagetrue
7729 \pgf@relevantforpicturesizefalse
7730 \@@_qpoint:n { row - #1 }
7731 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7732 \@@_qpoint:n { col - #2 }
7733 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7734 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7735 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7736 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7737 \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.

```

7738 \@@_pgf_rect_node:nnnnn
7739 { \@@_env: - #1 - #2 - block }

```

```

7740 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7741 \str_if_empty:NF \l_@@_block_name_str
7742 {
7743   \pgfnodealias
7744   { \@@_env: - \l_@@_block_name_str }
7745   { \@@_env: - #1 - #2 - block }
7746   \str_if_empty:NF \l_@@_name_str
7747   {
7748     \pgfnodealias
7749     { \l_@@_name_str - \l_@@_block_name_str }
7750     { \@@_env: - #1 - #2 - block }
7751   }
7752 }

```

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.

```

7753 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7754 {
7755   \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.

```

7756 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7757 {

```

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.

```

7758 \cs_if_exist:cT
7759 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7760 {
7761   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7762   {
7763     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7764     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7765   }
7766 }
7767 }

```

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.

```

7768 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7769 {
7770   \@@_qpoint:n { col - #2 }
7771   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7772 }
7773 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7774 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7775 {
7776   \cs_if_exist:cT
7777   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7778   {
7779     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7780     {
7781       \pgfpointanchor
7782       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7783       { east }
7784       \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7785     }
7786   }
7787 }
7788 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7789 {

```

```

7790     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7791     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7792   }
7793   \l_@@_pgf_rect_node:nnnnn
7794   { \l_@@_env: - #1 - #2 - block - short }
7795   \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7796 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\l_@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

7797 \bool_if:NT \l_@@_medium_nodes_bool
7798 {
7799   \l_@@_pgf_rect_node:nnn
7800   { \l_@@_env: - #1 - #2 - block - medium }
7801   { \pgfpointanchor { \l_@@_env: - #1 - #2 - medium } { north-west } }
7802   {
7803     \pgfpointanchor
7804     { \l_@@_env:
7805       - \int_use:N \l_@@_last_row_int
7806       - \int_use:N \l_@@_last_col_int - medium
7807     }
7808     { south-east }
7809   }
7810 }
7811 \endpgfpicture

```

```

7812 \bool_if:NTF \l_@@_ampersand_bool
7813 {
7814   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7815   \int_zero_new:N \l_@@_split_int
7816   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7817   \pgfpicture
7818   \pgfrememberpicturepositiononpagetrue
7819   \pgf@relevantforpicturesizefalse
7820
7821   \l_@@_qpoint:n { row - #1 }
7822   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7823   \l_@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7824   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7825   \l_@@_qpoint:n { col - #2 }
7826   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7827   \l_@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7828   \dim_set:Nn \l_tmpb_dim
7829   { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7830   \bool_lazy_or:nnT
7831   \l_@@_vlines_block_bool
7832   { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7833   {
7834     \int_step_inline:nn { \l_@@_split_int - 1 }
7835     {
7836       \pgfpathmoveto
7837       {
7838         \pgfpoint
7839         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7840         \l_@@_tmpc_dim
7841       }
7842       \pgfpathlineto
7843       {
7844         \pgfpoint
7845         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7846         \l_@@_tmpd_dim
7847       }
7848       \CT@arc@

```

```

7849         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7850         \pgfsetrectcap
7851         \pgfusepathqstroke
7852     }
7853 }
7854 \@@_qpoint:n { row - #1 - base }
7855 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7856 \int_step_inline:nn \l_@@_split_int
7857 {
7858     \group_begin:
7859     \dim_set:Nn \col@sep
7860     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7861     \pgftransformshift
7862     {
7863         \pgfpoint
7864         {
7865             \l_tmpa_dim + ##1 \l_tmpb_dim -
7866             \str_case:on \l_@@_hpos_block_str
7867             {
7868                 l { \l_tmpb_dim + \col@sep }
7869                 c { 0.5 \l_tmpb_dim }
7870                 r { \col@sep }
7871             }
7872         }
7873         { \l_@@_tmpc_dim }
7874     }
7875     \pgfset { inner~sep = \c_zero_dim }
7876     \pgfnode
7877     { rectangle }
7878     {
7879         \str_case:on \l_@@_hpos_block_str
7880         {
7881             c { base }
7882             l { base~west }
7883             r { base~east }
7884         }
7885     }
7886     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7887     \group_end:
7888 }
7889 \endpgfpicture
7890 }

```

Now the case where there is no ampersand & in the content of the block.

```

7891 {
7892     \bool_if:NTF \l_@@_p_block_bool
7893     {

```

When the final user has used the key p, we have to compute the width.

```

7894 \pgfpicture
7895 \pgfrememberpicturepositiononpagetrue
7896 \pgf@relevantforpicturesizefalse
7897 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7898 {
7899     \@@_qpoint:n { col - #2 }
7900     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7901     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7902 }
7903 {
7904     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7905     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7906     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7907 }
7908 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }

```

```

7909 \endpgfpicture
7910 \hbox_set:Nn \l_@@_cell_box
7911 {
7912   \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7913     { \g_tmpb_dim }
7914     \str_case:on \l_@@_hpos_block_str
7915       { c \centering r \raggedleft l \raggedright j { } }
7916       #6
7917     \end { minipage }
7918   }
7919 }
7920 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7921 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Now, we will put the label of the block. We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7922 \pgfpicture
7923 \pgfrememberpicturepositiononpagetrue
7924 \pgf@relevantforpicturesizefalse
7925 \bool_lazy_any:nTF
7926 {
7927   { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7928   { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7929   { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7930   { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7931 }
7932 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7933 \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`.

```

7934 \bool_if:nT \g_@@_last_col_found_bool
7935 {
7936   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7937     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7938 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7939 \tl_set:Ne \l_tmpa_tl
7940 {
7941   \str_case:on \l_@@_vpos_block_str
7942   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7943 { } { % added 2024-06-29
7944   \str_case:on \l_@@_hpos_block_str
7945   {
7946     c { center }
7947     l { west }
7948     r { east }
7949     j { center }
7950   }
7951 }
7952 c {
7953   \str_case:on \l_@@_hpos_block_str
7954   {
7955     c { center }
7956     l { west }
7957     r { east }
7958     j { center }
7959   }

```



```

7960     }
7961     T {
7962         \str_case:on \l_@@_hpos_block_str
7963         {
7964             c { north }
7965             l { north-west }
7966             r { north-east }
7967             j { north }
7968         }
7969     }
7970
7971     }
7972     B {
7973         \str_case:on \l_@@_hpos_block_str
7974         {
7975             c { south }
7976             l { south-west }
7977             r { south-east }
7978             j { south }
7979         }
7980     }
7981
7982     }
7983 }
7984 \pgftransformshift
7985 {
7986     \pgfpointanchor
7987     {
7988         \@@_env: - #1 - #2 - block
7989         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7990     }
7991     { \l_tmpa_tl }
7992 }
7993 \pgfset { inner~sep = \c_zero_dim }
7994 \pgfnode
7995 { rectangle }
7996 { \l_tmpa_tl }
7997 { \box_use_drop:N \l_@@_cell_box } { } { }
7998 }

```

End of the case when `\l_@@_vpos_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7999 {
8000     \pgfextracty \l_tmpa_dim
8001     {
8002         \@@_qpoint:n
8003         {
8004             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
8005             - base
8006         }
8007     }
8008     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

8009     \pgfpointanchor
8010     {
8011         \@@_env: - #1 - #2 - block
8012         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
8013     }
8014     {
8015         \str_case:on \l_@@_hpos_block_str
8016         {
8017             c { center }
8018             l { west }

```

```

8019         r { east }
8020         j { center }
8021     }
8022 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

8023     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
8024     \pgfset { inner~sep = \c_zero_dim }
8025     \pgfnode
8026     { rectangle }
8027     {
8028         \str_case:on \l_@@_hpos_block_str
8029         {
8030             c { base }
8031             l { base~west }
8032             r { base~east }
8033             j { base }
8034         }
8035     }
8036     { \box_use_drop:N \l_@@_cell_box } { } { }
8037 }
8038 \endpgfpicture
8039 }
8040 \group_end:
8041 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

8042 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
8043 {
8044     \pgfpicture
8045     \pgfrememberpicturepositiononpagetrue
8046     \pgf@relevantforpicturesizefalse
8047     \pgfpathrectanglecorners
8048     { \pgfpoint { #2 } { #3 } }
8049     { \pgfpoint { #4 } { #5 } }
8050     \pgfsetfillcolor { #1 }
8051     \pgfusepath { fill }
8052     \endpgfpicture
8053 }

```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```

8054 \cs_new_protected:Npn \@@_add_opacity_to_fill:
8055 {
8056     \tl_if_empty:NF \l_@@_opacity_tl
8057     {
8058         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
8059             {
8060                 \tl_set:Ne \l_@@_fill_tl
8061                 {
8062                     [ opacity = \l_@@_opacity_tl ,
8063                     \tl_tail:o \l_@@_fill_tl
8064                 }
8065             }
8066             {
8067                 \tl_set:Ne \l_@@_fill_tl
8068                 { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
8069             }
8070         }
8071     }

```

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).

```

8072 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8073 {
8074   \group_begin:
8075   \tl_clear:N \l_@@_draw_tl
8076   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8077   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8078   \pgfpicture
8079   \pgfrememberpicturepositiononpagetrue
8080   \pgf@relevantforpicturesizefalse
8081   \tl_if_empty:NF \l_@@_draw_tl
8082   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8083     \tl_if_eq:NnTF \l_@@_draw_tl { default }
8084     { \CT@arc@ }
8085     { \@@_color:o \l_@@_draw_tl }
8086   }
8087   \pgfsetcornersarced
8088   {
8089     \pgfpoint
8090     { \l_@@_rounded_corners_dim }
8091     { \l_@@_rounded_corners_dim }
8092   }
8093   \@@_cut_on_hyphen:w #2 \q_stop
8094   \int_compare:nNnF \l_tmpa_tl > \c@iRow
8095   {
8096     \int_compare:nNnF \l_tmpb_tl > \c@jCol
8097     {
8098       \@@_qpoint:n { row - \l_tmpa_tl }
8099       \dim_set_eq:NN \l_tmpb_dim \pgf@y
8100       \@@_qpoint:n { col - \l_tmpb_tl }
8101       \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8102       \@@_cut_on_hyphen:w #3 \q_stop
8103       \int_compare:nNnT \l_tmpa_tl > \c@iRow
8104       { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8105       \int_compare:nNnT \l_tmpb_tl > \c@jCol
8106       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8107       \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8108       \dim_set_eq:NN \l_tmpa_dim \pgf@y
8109       \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8110       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8111       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8112       \pgfpathrectanglecorners
8113       { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8114       { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8115       \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8116       { \pgfusepathqstroke }
8117       { \pgfusepath { stroke } }
8118     }
8119   }
8120   \endpgfpicture
8121   \group_end:
8122 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8123 \keys_define:nn { nicematrix / BlockStroke }
8124 {
8125   color .tl_set:N = \l_@@_draw_tl ,
8126   draw .code:n =
8127     \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,

```

```

8128     draw .default:n = default ,
8129     line-width .dim_set:N = \l_@@_line_width_dim ,
8130     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8131     rounded-corners .default:n = 4 pt
8132 }

```

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).

```

8133 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8134 {
8135   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8136   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8137   \@@_cut_on_hyphen:w #2 \q_stop
8138   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8139   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8140   \@@_cut_on_hyphen:w #3 \q_stop
8141   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8142   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8143   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8144   {
8145     \use:e
8146     {
8147       \@@_vline:n
8148       {
8149         position = ##1 ,
8150         start = \l_@@_tmpc_tl ,
8151         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8152         total-width = \dim_use:N \l_@@_line_width_dim
8153       }
8154     }
8155   }
8156 }
8157 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8158 {
8159   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8160   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8161   \@@_cut_on_hyphen:w #2 \q_stop
8162   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8163   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8164   \@@_cut_on_hyphen:w #3 \q_stop
8165   \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8166   \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8167   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8168   {
8169     \use:e
8170     {
8171       \@@_hline:n
8172       {
8173         position = ##1 ,
8174         start = \l_@@_tmpd_tl ,
8175         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8176         total-width = \dim_use:N \l_@@_line_width_dim
8177       }
8178     }
8179   }
8180 }

```

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).

```

8181 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3

```

```

8182 {
8183   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8184   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8185   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8186     { \@@_error:n { borders~forbidden } }
8187     {
8188       \tl_clear_new:N \l_@@_borders_tikz_tl
8189       \keys_set:no
8190         { nicematrix / OnlyForTikzInBorders }
8191         \l_@@_borders_clist
8192         \@@_cut_on_hyphen:w #2 \q_stop
8193         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8194         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8195         \@@_cut_on_hyphen:w #3 \q_stop
8196         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8197         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8198         \@@_stroke_borders_block_i:
8199       }
8200     }
8201   \hook_gput_code:nnn { begindocument } { . }
8202   {
8203     \cs_new_protected:Npe \@@_stroke_borders_block_i:
8204       {
8205         \c_@@_pgfortikzpicture_tl
8206         \@@_stroke_borders_block_ii:
8207         \c_@@_endpgfortikzpicture_tl
8208       }
8209   }
8210   \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8211     {
8212       \pgfrememberpicturepositiononpagetrue
8213       \pgf@relevantforpicturesizefalse
8214       \CT@arc@
8215       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8216       \clist_if_in:NnT \l_@@_borders_clist { right }
8217         { \@@_stroke_vertical:n \l_tmpb_tl }
8218       \clist_if_in:NnT \l_@@_borders_clist { left }
8219         { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8220       \clist_if_in:NnT \l_@@_borders_clist { bottom }
8221         { \@@_stroke_horizontal:n \l_tmpa_tl }
8222       \clist_if_in:NnT \l_@@_borders_clist { top }
8223         { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8224     }
8225   \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8226     {
8227       tikz .code:n =
8228         \cs_if_exist:NTF \tikzpicture
8229           { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8230           { \@@_error:n { tikz~in~borders~without~tikz } } ,
8231       tikz .value_required:n = true ,
8232       top .code:n = ,
8233       bottom .code:n = ,
8234       left .code:n = ,
8235       right .code:n = ,
8236       unknown .code:n = \@@_error:n { bad~border }
8237     }

```

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).

```

8238 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8239 {
8240   \@@_qpoint:n \l_@@_tmpc_tl

```

```

8241 \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8242 \@@_qpoint:n \l_tmpa_tl
8243 \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8244 \@@_qpoint:n { #1 }
8245 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8246 {
8247   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8248   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8249   \pgfusepathqstroke
8250 }
8251 {
8252   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8253     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8254 }
8255 }

```

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).

```

8256 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8257 {
8258   \@@_qpoint:n \l_@@_tmpd_tl
8259   \clist_if_in:NnTF \l_@@_borders_clist { left }
8260     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8261     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8262   \@@_qpoint:n \l_tmpb_tl
8263   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8264   \@@_qpoint:n { #1 }
8265   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8266   {
8267     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8268     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8269     \pgfusepathqstroke
8270   }
8271   {
8272     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8273       ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8274   }
8275 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8276 \keys_define:nn { nicematrix / BlockBorders }
8277 {
8278   borders .clist_set:N = \l_@@_borders_clist ,
8279   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8280   rounded-corners .default:n = 4 pt ,
8281   line-width .dim_set:N = \l_@@_line_width_dim
8282 }

```

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.

```

8283 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8284 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8285 {
8286   \begin { tikzpicture }
8287     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because #5 is a list of lists.

```
8288 \clist_map_inline:nn { #1 }
8289 {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```
8290 \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8291 \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8292 (
8293 [
8294 xshift = \dim_use:N \l_@@_offset_dim ,
8295 yshift = - \dim_use:N \l_@@_offset_dim
8296 ]
8297 #2 -| #3
8298 )
8299 rectangle
8300 (
8301 [
8302 xshift = - \dim_use:N \l_@@_offset_dim ,
8303 yshift = \dim_use:N \l_@@_offset_dim
8304 ]
8305 \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8306 ) ;
8307 }
8308 \end { tikzpicture }
8309 }

8310 \keys_define:nn { nicematrix / SpecialOffset }
8311 { 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.

```
8312 \cs_new_protected:Npn \@@_NullBlock:
8313 { \@@_collect_options:n { \@@_NullBlock_i: } }
8314 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8315 { }
```

27 How to draw the dotted lines transparently

```
8316 \cs_set_protected:Npn \@@_renew_matrix:
8317 {
8318 \RenewDocumentEnvironment { pmatrix } { } {
8319 { \pNiceMatrix }
8320 { \endpNiceMatrix }
8321 \RenewDocumentEnvironment { vmatrix } { } {
8322 { \vNiceMatrix }
8323 { \endvNiceMatrix }
8324 \RenewDocumentEnvironment { Vmatrix } { } {
8325 { \VNiceMatrix }
8326 { \endVNiceMatrix }
8327 \RenewDocumentEnvironment { bmatrix } { } {
8328 { \bNiceMatrix }
8329 { \endbNiceMatrix }
8330 \RenewDocumentEnvironment { Bmatrix } { } {
8331 { \BNiceMatrix }
8332 { \endBNiceMatrix }
8333 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8334 \keys_define:nn { nicematrix / Auto }
8335 {
8336   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8337   columns-type .value_required:n = true ,
8338   l .meta:n = { columns-type = l } ,
8339   r .meta:n = { columns-type = r } ,
8340   c .meta:n = { columns-type = c } ,
8341   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8342   delimiters / color .value_required:n = true ,
8343   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8344   delimiters / max-width .default:n = true ,
8345   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8346   delimiters .value_required:n = true ,
8347   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8348   rounded-corners .default:n = 4 pt
8349 }

8350 \NewDocumentCommand \AutoNiceMatrixWithDelims
8351 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8352 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8353 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8354 {

```

The group is for the protection of the keys.

```

8355   \group_begin:
8356   \keys_set:known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8357   \use:e
8358   {
8359     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8360     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8361     [ \exp_not:o \l_tmpa_tl ]
8362   }
8363   \int_if_zero:nT \l_@@_first_row_int
8364   {
8365     \int_if_zero:nT \l_@@_first_col_int { & }
8366     \prg_replicate:nn { #4 - 1 } { & }
8367     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8368   }
8369   \prg_replicate:nn { #3 }
8370   {
8371     \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`).

```

8372     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8373     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8374   }
8375   \int_compare:nNnT \l_@@_last_row_int > { -2 }
8376   {
8377     \int_if_zero:nT \l_@@_first_col_int { & }
8378     \prg_replicate:nn { #4 - 1 } { & }
8379     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8380   }
8381   \end { NiceArrayWithDelims }
8382   \group_end:
8383 }

8384 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8385 {

```



```

8386 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8387 {
8388   \bool_gset_true:N \g_@@_delims_bool
8389   \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8390   \AutoNiceMatrixWithDelims { #2 } { #3 }
8391 }
8392 }

8393 \@@_define_com:nnn p ( )
8394 \@@_define_com:nnn b [ ]
8395 \@@_define_com:nnn v | |
8396 \@@_define_com:nnn V \l \l
8397 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8398 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8399 {
8400   \group_begin:
8401   \bool_gset_false:N \g_@@_delims_bool
8402   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8403   \group_end:
8404 }

```

29 The redefinition of the command `\dotfill`

```

8405 \cs_set_eq:NN \@@_old_dotfill \dotfill
8406 \cs_new_protected:Npn \@@_dotfill:
8407 {

```

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}`).

```

8408   \@@_old_dotfill
8409   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8410 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8411 \cs_new_protected:Npn \@@_dotfill_i:
8412 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

30 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8413 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8414 {
8415   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8416   {
8417     \@@_actually_diagbox:nnnnnn
8418     { \int_use:N \c@iRow }
8419     { \int_use:N \c@jCol }
8420     { \int_use:N \c@iRow }
8421     { \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.

```
8422         { \g_@@_row_style_tl \exp_not:n { #1 } }
8423         { \g_@@_row_style_tl \exp_not:n { #2 } }
8424     }
```

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`.

```
8425     \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8426     {
8427         { \int_use:N \c@iRow }
8428         { \int_use:N \c@jCol }
8429         { \int_use:N \c@iRow }
8430         { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8431         { }
8432     }
8433 }
```

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.

```
8434 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8435 {
8436     \pgfpicture
8437     \pgf@relevantforpicturesizefalse
8438     \pgfrememberpicturepositiononpagetrue
8439     \@@_qpoint:n { row - #1 }
8440     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8441     \@@_qpoint:n { col - #2 }
8442     \dim_set_eq:NN \l_tmpb_dim \pgf@x
8443     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8444     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8445     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8446     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8447     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8448     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8449 }
```

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.

```
8450     \CT@arc@
8451     \pgfsetroundcap
8452     \pgfusepathqstroke
8453 }
8454 \pgfset { inner~sep = 1 pt }
8455 \pgfscope
8456 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8457 \pgfnode { rectangle } { south~west }
8458 {
8459     \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```
8460     \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8461     \end { minipage }
8462 }
8463 { }
8464 { }
```

```

8465 \endpgfscope
8466 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8467 \pgfnode { rectangle } { north~east }
8468 {
8469   \begin { minipage } { 20 cm }
8470   \raggedleft
8471   \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8472   \end { minipage }
8473 }
8474 { }
8475 { }
8476 \endpgfpicture
8477 }

```

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`.

```

8478 \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 `\`.

```

8479 \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`.

```

8480 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8481 {
8482   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8483   \@@_CodeAfter_iv:n
8484 }

```

We catch the argument of the command `\end` (in `#1`).

```

8485 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8486 {

```

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.

```

8487   \str_if_eq:eeTF \@currenvir { #1 }
8488   { \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`.

```

8489   {
8490     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8491     \@@_CodeAfter_ii:n
8492   }
8493 }

```

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.

```

8494 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8495 {
8496   \pgfpicture
8497   \pgfrememberpicturepositiononpagetrue
8498   \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.

```

8499   \@@_qpoint:n { row - 1 }
8500   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8501   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8502   \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).

```

8503   \bool_if:nTF { #3 }
8504   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8505   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8506   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8507   {
8508     \cs_if_exist:cT
8509     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8510     {
8511       \pgfpointanchor
8512       { \@@_env: - ##1 - #2 }
8513       { \bool_if:nTF { #3 } { west } { east } }
8514       \dim_set:Nn \l_tmpa_dim
8515       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8516     }
8517   }

```

Now we can put the delimiter with a node of PGF.

```

8518   \pgfset { inner~sep = \c_zero_dim }
8519   \dim_zero:N \nulldelimiterspace
8520   \pgftransformshift
8521   {
8522     \pgfpoint
8523     { \l_tmpa_dim }
8524     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8525   }
8526   \pgfnode
8527   { rectangle }
8528   { \bool_if:nTF { #3 } { east } { west } }
8529   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8530   \nullfont
8531   \c_math_toggle_token
8532   \@@_color:o \l_@@_delimiters_color_tl
8533   \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

8534     \vcenter
8535     {
8536         \nullfont
8537         \hrule \@height
8538             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8539             \@depth \c_zero_dim
8540             \@width \c_zero_dim
8541     }
8542     \bool_if:nTF { #3 } { \right . } { \right #1 }
8543     \c_math_toggle_token
8544 }
8545 { }
8546 { }
8547 \endpgfpicture
8548 }

```

33 The command \SubMatrix

```

8549 \keys_define:nn { nicematrix / sub-matrix }
8550 {
8551     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8552     extra-height .value_required:n = true ,
8553     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8554     left-xshift .value_required:n = true ,
8555     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8556     right-xshift .value_required:n = true ,
8557     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8558     xshift .value_required:n = true ,
8559     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8560     delimiters / color .value_required:n = true ,
8561     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8562     slim .default:n = true ,
8563     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8564     hlines .default:n = all ,
8565     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8566     vlines .default:n = all ,
8567     hvlines .meta:n = { hlines, vlines } ,
8568     hvlines .value_forbidden:n = true
8569 }
8570 \keys_define:nn { nicematrix }
8571 {
8572     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8573     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8574     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8575     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8576 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8577 \keys_define:nn { nicematrix / SubMatrix }
8578 {
8579     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8580     delimiters / color .value_required:n = true ,
8581     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8582     hlines .default:n = all ,
8583     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8584     vlines .default:n = all ,
8585     hvlines .meta:n = { hlines, vlines } ,
8586     hvlines .value_forbidden:n = true ,
8587     name .code:n =

```

```

8588 \tl_if_empty:nTF { #1 }
8589 { \@@_error:n { Invalid-name } }
8590 {
8591   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8592   {
8593     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8594     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8595     {
8596       \str_set:Nn \l_@@_submatrix_name_str { #1 }
8597       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8598     }
8599   }
8600   { \@@_error:n { Invalid-name } }
8601 } ,
8602 name .value_required:n = true ,
8603 rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8604 rules .value_required:n = true ,
8605 code .tl_set:N = \l_@@_code_tl ,
8606 code .value_required:n = true ,
8607 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8608 }

8609 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8610 {
8611   \peek_remove_spaces:n
8612   {
8613     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8614     {
8615       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8616       [
8617         delimiters / color = \l_@@_delimiters_color_tl ,
8618         hlines = \l_@@_submatrix_hlines_clist ,
8619         vlines = \l_@@_submatrix_vlines_clist ,
8620         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8621         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8622         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8623         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8624         #5
8625       ]
8626     }
8627     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8628   }
8629 }

8630 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8631 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8632 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8633 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8634 {
8635   \seq_gput_right:Ne \g_@@_submatrix_seq
8636   {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8637   { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8638   { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8639   { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8640   { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8641 }
8642 }

```

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`.

```

8643 \hook_gput_code:nnn { begindocument } { . }
8644 {
8645   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8646   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8647   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8648     {
8649       \peek_remove_spaces:n
8650       {
8651         \@@_sub_matrix:nnnnnnn
8652         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8653       }
8654     }
8655 }

```

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).

```

8656 \NewDocumentCommand \@@_compute_i_j:nn
8657 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8658 { \@@_compute_i_j:nnnn #1 #2 }
8659 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8660 {
8661   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8662   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8663   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8664   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8665   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8666     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8667   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8668     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8669   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8670     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8671   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8672     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8673 }
8674 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8675 {
8676   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8677 \@@_compute_i_j:nn { #2 } { #3 }
8678 \int_compare:NnNT \l_@@_first_i_tl = \l_@@_last_i_tl
8679   { \cs_set_nopar:Npn \arraystretch { 1 } }
8680 \bool_lazy_or:nnTF
8681   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8682   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8683   { \@@_error:nn { Construct~too-large } { \SubMatrix } }
8684   {
8685     \str_clear_new:N \l_@@_submatrix_name_str
8686     \keys_set:nn { nicematrix / SubMatrix } { #5 }

```

```

8687 \pgfpicture
8688 \pgfrememberpicturepositiononpagetrue
8689 \pgf@relevantforpicturesizefalse
8690 \pgfset { inner~sep = \c_zero_dim }
8691 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8692 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

The last value of \int_step_inline:nnn is provided by currification.

8693 \bool_if:NTF \l_@@_submatrix_slim_bool
8694 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8695 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8696 {
8697   \cs_if_exist:cT
8698   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8699   {
8700     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8701     \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8702     { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8703   }
8704   \cs_if_exist:cT
8705   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8706   {
8707     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8708     \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8709     { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8710   }
8711 }
8712 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8713 { \@@_error:nn { Impossible~delimiter } { left } }
8714 {
8715   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8716   { \@@_error:nn { Impossible~delimiter } { right } }
8717   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8718 }
8719 \endpgfpicture
8720 }
8721 \group_end:
8722 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8723 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8724 {
8725   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8726   \dim_set:Nn \l_@@_y_initial_dim
8727   {
8728     \fp_to_dim:n
8729     {
8730       \pgf@y
8731       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8732     }
8733   }
8734   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8735   \dim_set:Nn \l_@@_y_final_dim
8736   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8737   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8738   {
8739     \cs_if_exist:cT
8740     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8741     {
8742       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8743       \dim_set:Nn \l_@@_y_initial_dim
8744       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8745     }

```



```

8746 \cs_if_exist:cT
8747 { \pgf @ sh @ ns @ \l_@@_env: - \l_@@_last_i_tl - ##1 }
8748 {
8749 \pgfpointanchor { \l_@@_env: - \l_@@_last_i_tl - ##1 } { south }
8750 \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8751 { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8752 }
8753 }
8754 \dim_set:Nn \l_tmpa_dim
8755 {
8756 \l_@@_y_initial_dim - \l_@@_y_final_dim +
8757 \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8758 }
8759 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8760 \group_begin:
8761 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8762 \l_@@_set_CT@arc@:o \l_@@_rules_color_tl
8763 \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`.

```

8764 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8765 {
8766 \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8767 {
8768 \int_compare:nNnT
8769 { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8770 {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8771 \l_@@_qpoint:n { col - ##1 }
8772 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8773 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8774 \pgfusepathqstroke
8775 }
8776 }
8777 }

```

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.

```

8778 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8779 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8780 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8781 {
8782 \bool_lazy_and:nnTF
8783 { \int_compare_p:nNn { ##1 } > \c_zero_int }
8784 {
8785 \int_compare_p:nNn
8786 { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8787 {
8788 \l_@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8789 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8790 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8791 \pgfusepathqstroke
8792 }
8793 { \l_@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8794 }

```

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.

```

8795 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8796 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8797 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8798 {
8799   \bool_lazy_and:nnTF
8800   { \int_compare_p:nNn { ##1 } > \c_zero_int }
8801   {
8802     \int_compare_p:nNn
8803     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8804   {
8805     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8806   \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8807   \dim_set:Nn \l_tmpa_dim
8808   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8809   \str_case:nn { #1 }
8810   {
8811     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8812     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8813     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8814   }
8815   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8816   \dim_set:Nn \l_tmpb_dim
8817   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8818   \str_case:nn { #2 }
8819   {
8820     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8821     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8822     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8823   }
8824   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8825   \pgfusepathqstroke
8826   \group_end:
8827 }
8828 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8829 }

```

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).

```

8830 \str_if_empty:NF \l_@@_submatrix_name_str
8831 {
8832   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8833   \l_@@_x_initial_dim \l_@@_y_initial_dim
8834   \l_@@_x_final_dim \l_@@_y_final_dim
8835 }
8836 \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`.

```

8837 \begin { pgfscope }
8838 \pgftransformshift
8839 {
8840   \pgfpoint
8841   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8842   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8843 }
8844 \str_if_empty:NTF \l_@@_submatrix_name_str
8845 { \@@_node_left:nn #1 { } }

```

```

8846     { \l_@@_node_left:nn #1 { \l_@@_env: - \l_@@_submatrix_name_str - left } }
8847 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8848 \pgftransformshift
8849 {
8850   \pgfpoint
8851   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8852   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8853 }
8854 \str_if_empty:NTF \l_@@_submatrix_name_str
8855 { \l_@@_node_right:nnnn #2 { } { #3 } { #4 } }
8856 {
8857   \l_@@_node_right:nnnn #2
8858   { \l_@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8859 }
8860 \cs_set_eq:NN \pgfpointanchor \l_@@_pgfpointanchor:n
8861 \flag_clear_new:N \l_@@_code_flag
8862 \l_@@_code_tl
8863 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8864 \cs_set_eq:NN \l_@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\l_@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

8865 \cs_new_protected:Npn \l_@@_pgfpointanchor:n #1
8866 {
8867   \use:e
8868   { \exp_not:N \l_@@_old_pgfpointanchor { \l_@@_pgfpointanchor_i:nn #1 } }
8869 }

```

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 -.

```

8870 \cs_new:Npn \l_@@_pgfpointanchor_i:nn #1 #2
8871 { #1 { \l_@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8872 \tl_const:Nn \c_@@_integers_alist_tl
8873 {
8874   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8875   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8876   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8877   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8878 }
8879 \cs_new:Npn \l_@@_pgfpointanchor_ii:w #1-#2\q_stop
8880 {

```

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`.

```

8881 \tl_if_empty:nTF { #2 }
8882 {
8883   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8884   {
8885     \flag_raise:N \l_@@_code_flag
8886     \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8887     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8888     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8889   }
8890   { #1 }
8891 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8892 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8893 }

```

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`).

```

8894 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8895 {
8896   \str_case:nnF { #1 }
8897   {
8898     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8899     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8900   }

```

Now the case of a node of the form $i-j$.

```

8901 {
8902   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8903   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8904 }
8905 }

```

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`).

```

8906 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8907 {
8908   \pgfnode
8909   { rectangle }
8910   { east }
8911   {
8912     \nullfont
8913     \c_math_toggle_token
8914     \@@_color:o \l_@@_delimiters_color_tl
8915     \left #1
8916     \vcenter
8917     {
8918       \nullfont
8919       \hrule \@height \l_tmpa_dim
8920       \@depth \c_zero_dim
8921       \@width \c_zero_dim
8922     }
8923     \right .
8924     \c_math_toggle_token
8925   }
8926   { #2 }

```

```

8927     { }
8928 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

8929 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8930 {
8931   \pgfnode
8932     { rectangle }
8933     { west }
8934     {
8935       \nullfont
8936       \c_math_toggle_token
8937       \colorlet { current-color } { . }
8938       \@@_color:o \l_@@_delimiters_color_tl
8939       \left .
8940       \vcenter
8941         {
8942           \nullfont
8943           \hrule \@height \l_tmpa_dim
8944             \@depth \c_zero_dim
8945             \@width \c_zero_dim
8946         }
8947       \right #1
8948       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8949       ^ { \color { current-color } \smash { #4 } }
8950       \c_math_toggle_token
8951     }
8952   { #2 }
8953   { }
8954 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8955 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8956 {
8957   \peek_remove_spaces:n
8958   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8959 }
8960 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8961 {
8962   \peek_remove_spaces:n
8963   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8964 }
8965 \keys_define:nn { nicematrix / Brace }
8966 {
8967   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8968   left-shorten .default:n = true ,
8969   left-shorten .value_forbidden:n = true ,
8970   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8971   right-shorten .default:n = true ,
8972   right-shorten .value_forbidden:n = true ,
8973   shorten .meta:n = { left-shorten , right-shorten } ,
8974   shorten .value_forbidden:n = true ,
8975   yshift .dim_set:N = \l_@@_brace_yshift_dim ,

```

```

8976 yshift .value_required:n = true ,
8977 yshift .initial:n = \c_zero_dim ,
8978 color .tl_set:N = \l_tmpa_tl ,
8979 color .value_required:n = true ,
8980 unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8981 }

```

#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*.

```

8982 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8983 {
8984   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8985   \@@_compute_i_j:nn { #1 } { #2 }
8986   \bool_lazy_or:nnTF
8987   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8988   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8989   {
8990     \str_if_eq:eeTF { #5 } { under }
8991     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8992     { \@@_error:nn { Construct~too~large } { \OverBrace } }
8993   }
8994   {
8995     \tl_clear:N \l_tmpa_tl
8996     \keys_set:nn { nicematrix / Brace } { #4 }
8997     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8998     \pgfpicture
8999     \pgfrememberpicturepositiononpagetrue
9000     \pgf@relevantforpicturesizefalse
9001     \bool_if:NT \l_@@_brace_left_shorten_bool
9002     {
9003       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
9004       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9005       {
9006         \cs_if_exist:cT
9007         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
9008         {
9009           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
9010
9011           \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
9012           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
9013         }
9014       }
9015     }
9016     \bool_lazy_or:nnT
9017     { \bool_not_p:n \l_@@_brace_left_shorten_bool }
9018     { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
9019     {
9020       \@@_qpoint:n { col - \l_@@_first_j_tl }
9021       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
9022     }
9023     \bool_if:NT \l_@@_brace_right_shorten_bool
9024     {
9025       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
9026       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
9027       {
9028         \cs_if_exist:cT
9029         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
9030         {
9031           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
9032           \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
9033           { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }

```

```

9034     }
9035   }
9036 }
9037 \bool_lazy_or:nnT
9038 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
9039 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
9040 {
9041   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
9042   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
9043 }
9044 \pgfset { inner~sep = \c_zero_dim }
9045 \str_if_eq:eeTF { #5 } { under }
9046 { \@@_underbrace_i:n { #3 } }
9047 { \@@_overbrace_i:n { #3 } }
9048 \endpgfpicture
9049 }
9050 \group_end:
9051 }

```

The argument is the text to put above the brace.

```

9052 \cs_new_protected:Npn \@@_overbrace_i:n #1
9053 {
9054   \@@_qpoint:n { row - \l_@@_first_i_tl }
9055   \pgftransformshift
9056   {
9057     \pgfpoint
9058     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9059     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
9060   }
9061   \pgfnode
9062   { rectangle }
9063   { south }
9064   {
9065     \vtop
9066     {
9067       \group_begin:
9068       \everycr { }
9069       \halign
9070       {
9071         \hfil ## \hfil \crcr
9072         \bool_if:NTF \l_@@_tabular_bool
9073         { \begin { tabular } { c } #1 \end { tabular } }
9074         { $ \begin { array } { c } #1 \end { array } $ }
9075         \cr
9076         \c_math_toggle_token
9077         \overbrace
9078         {
9079           \hbox_to_wd:nn
9080           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9081           { }
9082         }
9083         \c_math_toggle_token
9084         \cr
9085       }
9086       \group_end:
9087     }
9088   }
9089   { }
9090   { }
9091 }

```

The argument is the text to put under the brace.

```

9092 \cs_new_protected:Npn \@@_underbrace_i:n #1

```

```

9093 {
9094   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9095   \pgftransformshift
9096   {
9097     \pgfpoint
9098     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9099     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9100   }
9101   \pgfnode
9102   { rectangle }
9103   { north }
9104   {
9105     \group_begin:
9106     \everycr { }
9107     \vbox
9108     {
9109       \halign
9110       {
9111         \hfil ## \hfil \crcr
9112         \c_math_toggle_token
9113         \underbrace
9114         {
9115           \hbox_to_wd:nn
9116           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9117           { }
9118         }
9119         \c_math_toggle_token
9120         \cr
9121         \bool_if:NTF \l_@@_tabular_bool
9122         { \begin { tabular } { c } #1 \end { tabular } }
9123         { $ \begin { array } { c } #1 \end { array } $ }
9124         \cr
9125       }
9126     }
9127     \group_end:
9128   }
9129   { }
9130   { }
9131 }

```

35 The command TikzEveryCell

```

9132 \bool_new:N \l_@@_not_empty_bool
9133 \bool_new:N \l_@@_empty_bool
9134
9135 \keys_define:nn { nicematrix / TikzEveryCell }
9136 {
9137   not-empty .code:n =
9138     \bool_lazy_or:nnTF
9139     \l_@@_in_code_after_bool
9140     \g_@@_recreate_cell_nodes_bool
9141     { \bool_set_true:N \l_@@_not_empty_bool }
9142     { \@@_error:n { detection-of-empty-cells } } ,
9143   not-empty .value_forbidden:n = true ,
9144   empty .code:n =
9145     \bool_lazy_or:nnTF
9146     \l_@@_in_code_after_bool
9147     \g_@@_recreate_cell_nodes_bool
9148     { \bool_set_true:N \l_@@_empty_bool }

```



```

9149     { \@@_error:n { detection~of~empty~cells } } ,
9150     empty .value_forbidden:n = true ,
9151     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9152 }
9153
9154
9155 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9156 {
9157     \IfPackageLoadedTF { tikz }
9158     {
9159         \group_begin:
9160         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnn` is a *list of lists* of TikZ keys.

```

9161     \tl_set:Nn \l_tmpa_tl { { #2 } }
9162     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9163     { \@@_for_a_block:nnnn ##1 }
9164     \@@_all_the_cells:
9165     \group_end:
9166 }
9167 { \@@_error:n { TikzEveryCell~without~tikz } }
9168 }
9169
9170 \tl_new:N \@@_i_tl
9171 \tl_new:N \@@_j_tl
9172
9173
9174 \cs_new_protected:Nn \@@_all_the_cells:
9175 {
9176     \int_step_variable:nNn \c@iRow \@@_i_tl
9177     {
9178         \int_step_variable:nNn \c@jCol \@@_j_tl
9179         {
9180             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9181             {
9182                 \clist_if_in:NcF \l_@@_corners_cells_clist
9183                 { \@@_i_tl - \@@_j_tl }
9184                 {
9185                     \bool_set_false:N \l_tmpa_bool
9186                     \cs_if_exist:cTF
9187                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9188                     {
9189                         \bool_if:NF \l_@@_empty_bool
9190                         { \bool_set_true:N \l_tmpa_bool }
9191                     }
9192                     {
9193                         \bool_if:NF \l_@@_not_empty_bool
9194                         { \bool_set_true:N \l_tmpa_bool }
9195                     }
9196                     \bool_if:NT \l_tmpa_bool
9197                     {
9198                         \@@_block_tikz:nnnn
9199                         \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9200                     }
9201                 }
9202             }
9203         }
9204     }
9205 }
9206
9207 \cs_new_protected:Nn \@@_for_a_block:nnnn
9208 {
9209     \bool_if:NF \l_@@_empty_bool

```

```

9210 {
9211   \@@_block_tikz:nnnn
9212   \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9213 }
9214 \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9215 }
9216
9217 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9218 {
9219   \int_step_inline:nnn { #1 } { #3 }
9220   {
9221     \int_step_inline:nnn { #2 } { #4 }
9222     { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
9223   }
9224 }

```

36 The command \ShowCellNames

```

9225 \NewDocumentCommand \@@_ShowCellNames { }
9226 {
9227   \bool_if:NT \l_@@_in_code_after_bool
9228   {
9229     \pgfpicture
9230     \pgfrememberpicturepositiononpagetrue
9231     \pgf@relevantforpicturesizefalse
9232     \pgfpathrectanglecorners
9233     { \@@_qpoint:n { 1 } }
9234     {
9235       \@@_qpoint:n
9236       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9237     }
9238     \pgfsetfillopacity { 0.75 }
9239     \pgfsetfillcolor { white }
9240     \pgfusepathqfill
9241     \endpgfpicture
9242   }
9243   \dim_gzero_new:N \g_@@_tmpc_dim
9244   \dim_gzero_new:N \g_@@_tmpd_dim
9245   \dim_gzero_new:N \g_@@_tmpe_dim
9246   \int_step_inline:nn \c@iRow
9247   {
9248     \bool_if:NTF \l_@@_in_code_after_bool
9249     {
9250       \pgfpicture
9251       \pgfrememberpicturepositiononpagetrue
9252       \pgf@relevantforpicturesizefalse
9253     }
9254     { \begin { pgfpicture } }
9255     \@@_qpoint:n { row - ##1 }
9256     \dim_set_eq:NN \l_tmpa_dim \pgf@y
9257     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9258     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9259     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9260     \bool_if:NTF \l_@@_in_code_after_bool
9261     { \endpgfpicture }
9262     { \end { pgfpicture } }
9263     \int_step_inline:nn \c@jCol
9264     {
9265       \hbox_set:Nn \l_tmpa_box
9266       {
9267         \normalfont \Large \sfamily \bfseries
9268         \bool_if:NTF \l_@@_in_code_after_bool

```

```

9269         { \color { red } }
9270         { \color { red ! 50 } }
9271     ##1 - ###1
9272 }
9273 \bool_if:NTF \l_@@_in_code_after_bool
9274 {
9275     \pgfpicture
9276     \pgfrememberpicturepositiononpagetrue
9277     \pgf@relevantforpicturesizefalse
9278 }
9279 { \begin { pgfpicture } }
9280 \@@_qpoint:n { col - ###1 }
9281 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9282 \@@_qpoint:n { col - \int_eval:n { ###1 + 1 } }
9283 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9284 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9285 \bool_if:NTF \l_@@_in_code_after_bool
9286 { \endpgfpicture }
9287 { \end { pgfpicture } }
9288 \fp_set:Nn \l_tmpa_fp
9289 {
9290     \fp_min:nn
9291     {
9292         \fp_min:nn
9293         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9294         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9295     }
9296     { 1.0 }
9297 }
9298 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9299 \pgfpicture
9300 \pgfrememberpicturepositiononpagetrue
9301 \pgf@relevantforpicturesizefalse
9302 \pgftransformshift
9303 {
9304     \pgfpoint
9305     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9306     { \dim_use:N \g_tmpa_dim }
9307 }
9308 \pgfnode
9309 { rectangle }
9310 { center }
9311 { \box_use:N \l_tmpa_box }
9312 { }
9313 { }
9314 \endpgfpicture
9315 }
9316 }
9317 }

```

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.

```

9318 \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.

```

9319 \bool_new:N \g_@@_footnote_bool
9320 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9321 {
9322   The~key~'\l_keys_key_str'~is~unknown. \\
9323   That~key~will~be~ignored. \\
9324   For~a~list~of~the~available~keys,~type~H~<return>.
9325 }
9326 {
9327   The~available~keys~are~(in~alphabetic~order):~
9328   footnote,~
9329   footnotehyper,~
9330   messages~for~Overleaf,~
9331   renew~dots,~and~
9332   renew~matrix.
9333 }
9334 \@@_msg_new:nn { no-test-for-array }
9335 {
9336   The~key~'no-test-for-array'~has~been~deprecated~and~will~be~
9337   deleted~in~a~future~version~of~nicematrix.
9338 }
9339 \keys_define:nn { nicematrix / Package }
9340 {
9341   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9342   renew~dots .value_forbidden:n = true ,
9343   renew~matrix .code:n = \@@_renew_matrix: ,
9344   renew~matrix .value_forbidden:n = true ,
9345   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9346   footnote .bool_set:N = \g_@@_footnote_bool ,
9347   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. We keep the following key only for compatibility but maybe we will delete it.

```

9348   no-test-for-array .code:n = \@@_warning:n { no-test-for-array } ,
9349   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9350 }
9351 \ProcessKeysOptions { nicematrix / Package }

9352 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9353 {
9354   You~can't~use~the~option~'footnote'~because~the~package~
9355   footnotehyper~has~already~been~loaded.~
9356   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9357   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9358   of~the~package~footnotehyper.\\
9359   The~package~footnote~won't~be~loaded.
9360 }
9361 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9362 {
9363   You~can't~use~the~option~'footnotehyper'~because~the~package~
9364   footnote~has~already~been~loaded.~
9365   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9366   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9367   of~the~package~footnote.\\
9368   The~package~footnotehyper~won't~be~loaded.
9369 }

9370 \bool_if:NT \g_@@_footnote_bool
9371 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9372 \IfClassLoadedTF { beamer }
9373 { \bool_set_false:N \g_@@_footnote_bool }
9374 {
9375   \IfPackageLoadedTF { footnotehyper }
9376   { \@@_error:n { footnote~with~footnotehyper~package } }
9377   { \usepackage { footnote } }
9378 }
9379 }

9380 \bool_if:NT \g_@@_footnotehyper_bool
9381 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9382 \IfClassLoadedTF { beamer }
9383 { \bool_set_false:N \g_@@_footnote_bool }
9384 {
9385   \IfPackageLoadedTF { footnote }
9386   { \@@_error:n { footnotehyper~with~footnote~package } }
9387   { \usepackage { footnotehyper } }
9388 }
9389 \bool_set_true:N \g_@@_footnote_bool
9390 }

```

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.

```

9391 \bool_new:N \l_@@_underscore_loaded_bool
9392 \IfPackageLoadedT { underscore }
9393 { \bool_set_true:N \l_@@_underscore_loaded_bool }

9394 \hook_gput_code:nnn { begindocument } { . }
9395 {
9396   \bool_if:NF \l_@@_underscore_loaded_bool
9397   {
9398     \IfPackageLoadedT { underscore }
9399     { \@@_error:n { underscore~after~nicematrix } }
9400   }
9401 }

```

39 Error messages of the package

```

9402 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9403 { \str_const:Nn \c_@@_available_keys_str { } }
9404 {
9405   \str_const:Nn \c_@@_available_keys_str
9406   { For~a~list~of~the~available~keys,~type~H~<return>. }
9407 }

9408 \seq_new:N \g_@@_types_of_matrix_seq

```

```

9409 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9410 {
9411   NiceMatrix ,
9412   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9413 }
9414 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9415 { \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`.

```

9416 \cs_new_protected:Npn \@@_error_too_much_cols:
9417 {
9418   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9419   { \@@_fatal:nn { too-much-cols-for-array } }
9420   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9421   { \@@_fatal:n { too-much-cols-for-matrix } }
9422   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9423   { \@@_fatal:n { too-much-cols-for-matrix } }
9424   \bool_if:NF \l_@@_last_col_without_value_bool
9425   { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9426 }

```

The following command must *not* be protected since it's used in an error message.

```

9427 \cs_new:Npn \@@_message_hdotsfor:
9428 {
9429   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9430   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9431 }
9432 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9433 {
9434   Incompatible-options.\\
9435   You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9436   The-output-will-not-be-reliable.
9437 }
9438 \@@_msg_new:nn { key~color~inside }
9439 {
9440   Key-deprecated.\\
9441   The-key~'color~inside'~(and~its~alias~'colortbl~like')~is~now~point~less~
9442   and~have~been~deprecated.\\
9443   You~won't~have~similar~message~till~the~end~of~the~document.
9444 }
9445 \@@_msg_new:nn { negative~weight }
9446 {
9447   Negative-weight.\\
9448   The-weight-of~the~'X'~columns~must~be~positive~and~you~have~used~
9449   the~value~'\int_use:N \l_@@_weight_int'.\\
9450   The~absolute~value~will~be~used.
9451 }
9452 \@@_msg_new:nn { last~col~not~used }
9453 {
9454   Column~not~used.\\
9455   The-key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9456   in~your~\@@_full_name_env:..~However,~you~can~go~on.
9457 }
9458 \@@_msg_new:nn { too-much~cols~for~matrix~with~last~col }
9459 {
9460   Too-much-columns.\\
9461   In~the~row~\int_eval:n { \c@iRow },~
9462   you~try~to~use~more~columns~

```

```

9463     than-allowed-by-your~\@@_full_name_env:.\@@_message_hdotsfor:\
9464     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9465     (plus-the-exterior-columns).~This-error-is-fatal.
9466   }
9467 \@@_msg_new:nn { too-much-cols-for-matrix }
9468 {
9469   Too-much-columns.\\
9470   In-the-row~\int_eval:n { \c@iRow },~
9471   you-try-to-use-more-columns-than-allowed-by-your~
9472   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
9473   number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
9474   columns)-is-fixed-by-the-LaTeX-counter~'MaxMatrixCols'.~
9475   Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9476   \token_to_str:N \setcounter\ to-change-that-value).~
9477   This-error-is-fatal.
9478 }
9479 \@@_msg_new:nn { too-much-cols-for-array }
9480 {
9481   Too-much-columns.\\
9482   In-the-row~\int_eval:n { \c@iRow },~
9483   ~you-try-to-use-more-columns-than-allowed-by-your~
9484   \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9485   \int_use:N \g_@@_static_num_of_col_int
9486   \bool_if:nT
9487     { \int_compare_p:nNn \l_@@_first_col_int = 0 || \g_@@_last_col_found_bool }
9488     { ~(plus-the-exterior-ones) }.~
9489   This-error-is-fatal.
9490 }
9491 \@@_msg_new:nn { columns-not-used }
9492 {
9493   Columns-not-used.\\
9494   The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9495   \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9496   The-columns-you-did-not-used-won't-be-created.\\
9497   You-won't-have-similar-error-message-till-the-end-of-the-document.
9498 }
9499 \@@_msg_new:nn { empty-preamble }
9500 {
9501   Empty-preamble.\\
9502   The-preamble-of-your~\@@_full_name_env:\ is-empty.\\
9503   This-error-is-fatal.
9504 }
9505 \@@_msg_new:nn { in-first-col }
9506 {
9507   Erroneous-use.\\
9508   You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9509   That-command-will-be-ignored.
9510 }
9511 \@@_msg_new:nn { in-last-col }
9512 {
9513   Erroneous-use.\\
9514   You-can't-use-the-command~#1 in-the-last-column~(exterior)~of-the-array.\\
9515   That-command-will-be-ignored.
9516 }
9517 \@@_msg_new:nn { in-first-row }
9518 {
9519   Erroneous-use.\\
9520   You-can't-use-the-command~#1 in-the-first-row~(number~0)~of-the-array.\\
9521   That-command-will-be-ignored.
9522 }

```

```

9523 \@@_msg_new:nn { in~last~row }
9524 {
9525   Erroneous~use.\\
9526   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9527   That~command~will~be~ignored.
9528 }
9529 \@@_msg_new:nn { TopRule~without~booktabs }
9530 {
9531   Erroneous~use.\\
9532   You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
9533   That~command~will~be~ignored.
9534 }
9535 \@@_msg_new:nn { TopRule~without~tikz }
9536 {
9537   Erroneous~use.\\
9538   You~can't~use~the~command~ #1 because~'tikz'~is~not~loaded.\\
9539   That~command~will~be~ignored.
9540 }
9541 \@@_msg_new:nn { caption~outside~float }
9542 {
9543   Key~caption~forbidden.\\
9544   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9545   environment.~This~key~will~be~ignored.
9546 }
9547 \@@_msg_new:nn { short~caption~without~caption }
9548 {
9549   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9550   However,~your~'short~caption'~will~be~used~as~'caption'.
9551 }
9552 \@@_msg_new:nn { double~closing~delimiter }
9553 {
9554   Double~delimiter.\\
9555   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9556   delimiter.~This~delimiter~will~be~ignored.
9557 }
9558 \@@_msg_new:nn { delimiter~after~opening }
9559 {
9560   Double~delimiter.\\
9561   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9562   delimiter.~That~delimiter~will~be~ignored.
9563 }
9564 \@@_msg_new:nn { bad~option~for~line~style }
9565 {
9566   Bad~line~style.\\
9567   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9568   is~'standard'.~That~key~will~be~ignored.
9569 }
9570 \@@_msg_new:nn { corners~with~no~cell~nodes }
9571 {
9572   Incompatible~keys.\\
9573   You~can't~use~the~key~'corners'~here~because~the~key~'no~cell~nodes'~
9574   is~in~force.\\
9575   If~you~go~on,~that~key~will~be~ignored.
9576 }
9577 \@@_msg_new:nn { extra~nodes~with~no~cell~nodes }
9578 {
9579   Incompatible~keys.\\
9580   You~can't~create~'extra~nodes'~here~because~the~key~'no~cell~nodes'~
9581   is~in~force.\\
9582   If~you~go~on,~those~extra~nodes~won't~be~created.

```



```

9583 }
9584 \@@_msg_new:nn { Identical-notes-in-caption }
9585 {
9586   Identical~tabular-notes.\\
9587   You~can't~put~several~notes~with~the~same~content~in~
9588   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9589   If~you~go~on,~the~output~will~probably~be~erroneous.
9590 }
9591 \@@_msg_new:nn { tabularnote~below~the~tabular }
9592 {
9593   \token_to_str:N \tabularnote\ forbidden\\
9594   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9595   of~your~tabular~because~the~caption~will~be~composed~below~
9596   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9597   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9598   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9599   no~similar~error~will~raised~in~this~document.
9600 }
9601 \@@_msg_new:nn { Unknown~key~for~rules }
9602 {
9603   Unknown~key.\\
9604   There~is~only~two~keys~available~here:~width~and~color.\\
9605   Your~key~'\l_keys_key_str'~will~be~ignored.
9606 }
9607 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9608 {
9609   Unknown~key.\\
9610   There~is~only~two~keys~available~here:~
9611   'empty'~and~'not~empty'.\\
9612   Your~key~'\l_keys_key_str'~will~be~ignored.
9613 }
9614 \@@_msg_new:nn { Unknown~key~for~rotate }
9615 {
9616   Unknown~key.\\
9617   The~only~key~available~here~is~'c'.\\
9618   Your~key~'\l_keys_key_str'~will~be~ignored.
9619 }
9620 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9621 {
9622   Unknown~key.\\
9623   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9624   It~you~go~on,~you~will~probably~have~other~errors. \\
9625   \c_@@_available_keys_str
9626 }
9627 {
9628   The~available~keys~are~(in~alphabetic~order):~
9629   ccommand,~
9630   color,~
9631   command,~
9632   dotted,~
9633   letter,~
9634   multiplicity,~
9635   sep-color,~
9636   tikz,~and~total~width.
9637 }
9638 \@@_msg_new:nnn { Unknown~key~for~xdots }
9639 {
9640   Unknown~key.\\
9641   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9642   \c_@@_available_keys_str
9643 }

```

```

9644 {
9645   The-available-keys-are-(in-alphabetic-order):~
9646   'color',~
9647   'horizontal-labels',~
9648   'inter',~
9649   'line-style',~
9650   'radius',~
9651   'shorten',~
9652   'shorten-end'~and~'shorten-start'.
9653 }
9654 \@@_msg_new:nn { Unknown-key-for-rowcolors }
9655 {
9656   Unknown-key.\\
9657   As-for-now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
9658   (and-you-try-to-use~'\l_keys_key_str')\\
9659   That-key-will-be-ignored.
9660 }
9661 \@@_msg_new:nn { label-without-caption }
9662 {
9663   You-can't-use-the-key~'label'~in-your~'{NiceTabular}'~because~
9664   you-have-not-used-the-key~'caption'.~The-key~'label'~will-be-ignored.
9665 }
9666 \@@_msg_new:nn { W-warning }
9667 {
9668   Line~\msg_line_number:.~The-cell-is-too-wide-for-your-column~'W'~
9669   (row~\int_use:N \c@iRow).
9670 }
9671 \@@_msg_new:nn { Construct-too-large }
9672 {
9673   Construct-too-large.\\
9674   Your-command~\token_to_str:N #1
9675   can't-be-drawn-because-your-matrix-is-too-small.\\
9676   That-command-will-be-ignored.
9677 }
9678 \@@_msg_new:nn { underscore-after-nicematrix }
9679 {
9680   Problem-with~'underscore'.\\
9681   The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9682   You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\\
9683   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9684 }
9685 \@@_msg_new:nn { ampersand-in-light-syntax }
9686 {
9687   Ampersand-forbidden.\\
9688   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9689   the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9690 }
9691 \@@_msg_new:nn { double-backslash-in-light-syntax }
9692 {
9693   Double-backslash-forbidden.\\
9694   You-can't-use~\token_to_str:N
9695   \\~to-separate-rows-because-the-key~'light-syntax'~
9696   is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9697   (set-by-the-key~'end-of-row').~This-error-is-fatal.
9698 }
9699 \@@_msg_new:nn { hlines-with-color }
9700 {
9701   Incompatible-keys.\\
9702   You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for-a~
9703   '\token_to_str:N \Block'~when-the-key~'color'~or~'draw'~is-used.\\

```

```

9704     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9705     Your~key~will~be~discarded.
9706 }
9707 \@@_msg_new:nn { bad~value~for~baseline }
9708 {
9709     Bad~value~for~baseline.\\
9710     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9711     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9712     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9713     the~form~'line-i'.\\
9714     A~value~of~1~will~be~used.
9715 }
9716 \@@_msg_new:nn { detection~of~empty~cells }
9717 {
9718     Problem~with~'not~empty'\\
9719     For~technical~reasons,~you~must~activate~
9720     'create~cell~nodes'~in~\token_to_str:N \CodeBefore\
9721     in~order~to~use~the~key~'\l_keys_key_str'.\\
9722     That~key~will~be~ignored.
9723 }
9724 \@@_msg_new:nn { siunitx~not~loaded }
9725 {
9726     siunitx~not~loaded\\
9727     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9728     That~error~is~fatal.
9729 }
9730 \@@_msg_new:nn { Invalid~name }
9731 {
9732     Invalid~name.\\
9733     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9734     \SubMatrix\ of~your~\@@_full_name_env:.\
9735     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9736     This~key~will~be~ignored.
9737 }
9738 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9739 {
9740     Wrong~line.\\
9741     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9742     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9743     number~is~not~valid.~It~will~be~ignored.
9744 }
9745 \@@_msg_new:nn { Impossible~delimiter }
9746 {
9747     Impossible~delimiter.\\
9748     It's~impossible~to~draw~the~#1~delimiter~of~your~
9749     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9750     in~that~column.
9751     \bool_if:NT \l_@@_submatrix_slim_bool
9752     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9753     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9754 }
9755 \@@_msg_new:nnn { width~without~X~columns }
9756 {
9757     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9758     That~key~will~be~ignored.
9759 }
9760 {
9761     This~message~is~the~message~'width~without~X~columns'~
9762     of~the~module~'nicematrix'.~
9763     The~experimented~users~can~disable~that~message~with~
9764     \token_to_str:N \msg_redirect_name:nnn.\\

```

```

9765 }
9766
9767 \@@_msg_new:nn { key-multiplicity-with-dotted }
9768 {
9769   Incompatible-keys. \\
9770   You-have-used-the-key~'multiplicity'~with~the-key~'dotted'~
9771   in~a~'custom-line'~.~They~are~incompatible. \\
9772   The-key~'multiplicity'~will~be~discarded.
9773 }
9774 \@@_msg_new:nn { empty-environment }
9775 {
9776   Empty-environment.\\
9777   Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9778 }
9779 \@@_msg_new:nn { No-letter-and-no-command }
9780 {
9781   Erroneous-use.\\
9782   Your-use-of~'custom-line'~is~no-op~since~you~don't~have~used~the~
9783   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9784   ~'ccommand'~(to~draw~horizontal~rules).\\
9785   However,~you~can~go~on.
9786 }
9787 \@@_msg_new:nn { Forbidden-letter }
9788 {
9789   Forbidden-letter.\\
9790   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9791   It~will~be~ignored.
9792 }
9793 \@@_msg_new:nn { Several-letters }
9794 {
9795   Wrong-name.\\
9796   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9797   have~used~'\l_@@_letter_str').\\
9798   It~will~be~ignored.
9799 }
9800 \@@_msg_new:nn { Delimiter-with-small }
9801 {
9802   Delimiter~forbidden.\\
9803   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9804   because~the~key~'small'~is~in~force.\\
9805   This-error-is-fatal.
9806 }
9807 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9808 {
9809   Unknown-cell.\\
9810   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9811   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9812   can't~be~executed~because~a~cell~doesn't~exist.\\
9813   This~command~\token_to_str:N \line\ will~be~ignored.
9814 }
9815 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9816 {
9817   Duplicate-name.\\
9818   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9819   in~this~\@@_full_name_env:.\\
9820   This~key~will~be~ignored.\\
9821   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9822     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9823 }
9824 {

```

```

9825     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9826     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9827 }

9828 \@@_msg_new:nn { r~or~l~with~preamble }
9829 {
9830     Erroneous~use.\\
9831     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9832     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9833     your~\@@_full_name_env:~
9834     This~key~will~be~ignored.
9835 }

9836 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9837 {
9838     Erroneous~use.\\
9839     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9840     the~array.~This~error~is~fatal.
9841 }

9842 \@@_msg_new:nn { bad~corner }
9843 {
9844     Bad~corner.\\
9845     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9846     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9847     This~specification~of~corner~will~be~ignored.
9848 }

9849 \@@_msg_new:nn { bad~border }
9850 {
9851     Bad~border.\\
9852     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9853     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9854     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9855     also~use~the~key~'tikz'
9856     \IfPackageLoadedF { tikz }
9857     {~if~you~load~the~LaTeX~package~'tikz'}).\\
9858     This~specification~of~border~will~be~ignored.
9859 }

9860 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9861 {
9862     TikZ~not~loaded.\\
9863     You~can't~use~\token_to_str:N \TikzEveryCell\
9864     because~you~have~not~loaded~tikz.~
9865     This~command~will~be~ignored.
9866 }

9867 \@@_msg_new:nn { tikz~key~without~tikz }
9868 {
9869     TikZ~not~loaded.\\
9870     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9871     \Block'~because~you~have~not~loaded~tikz.~
9872     This~key~will~be~ignored.
9873 }

9874 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9875 {
9876     Erroneous~use.\\
9877     In~the~\@@_full_name_env:,~you~must~use~the~key~
9878     'last~col'~without~value.\\
9879     However,~you~can~go~on~for~this~time~
9880     (the~value~'\l_keys_value_tl'~will~be~ignored).
9881 }

9882 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9883 {
9884     Erroneous~use.\\

```

```

9885     In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9886     'last-col'~without-value.\\
9887     However,~you-can-go-on-for~this-time~
9888     (the-value~'\l_keys_value_tl'~will-be-ignored).
9889 }
9890 \@@_msg_new:nn { Block-too-large-1 }
9891 {
9892     Block-too-large.\\
9893     You-try-to-draw-a-block-in-the-cell-#1-#2-of-your-matrix-but-the-matrix-is~
9894     too-small-for-that-block. \\
9895     This-block-and-maybe-others-will-be-ignored.
9896 }
9897 \@@_msg_new:nn { Block-too-large-2 }
9898 {
9899     Block-too-large.\\
9900     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9901     \g_@@_static_num_of_col_int\
9902     columns-but-you-use-only~\int_use:N \c@jCol\ and-that's-why-a-block~
9903     specified-in-the-cell-#1-#2-can't-be-drawn.~You-should-add-some-ampersands~
9904     (&)~at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9905     This-block-and-maybe-others-will-be-ignored.
9906 }
9907 \@@_msg_new:nn { unknown-column-type }
9908 {
9909     Bad-column-type.\\
9910     The-column-type~'#1'~in-your~\@@_full_name_env:\
9911     is-unknown. \\
9912     This-error-is-fatal.
9913 }
9914 \@@_msg_new:nn { unknown-column-type-S }
9915 {
9916     Bad-column-type.\\
9917     The-column-type~'S'~in-your~\@@_full_name_env:\ is-unknown. \\
9918     If-you-want-to-use-the-column-type~'S'~of-siunitx,~you-should~
9919     load-that-package. \\
9920     This-error-is-fatal.
9921 }
9922 \@@_msg_new:nn { tabularnote-forbidden }
9923 {
9924     Forbidden-command.\\
9925     You-can't-use-the-command~\token_to_str:N\ tabularnote\
9926     ~here.~This-command-is-available-only-in~
9927     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or-in~
9928     the-argument-of-a-command~\token_to_str:N \caption\ included~
9929     in-an-environment~{table}. \\
9930     This-command-will-be-ignored.
9931 }
9932 \@@_msg_new:nn { borders-forbidden }
9933 {
9934     Forbidden-key.\\
9935     You-can't-use-the-key~'borders'~of-the-command~\token_to_str:N \Block\
9936     because-the-option~'rounded-corners'~
9937     is-in-force-with-a-non-zero-value.\\
9938     This-key-will-be-ignored.
9939 }
9940 \@@_msg_new:nn { bottomrule-without-booktabs }
9941 {
9942     booktabs-not-loaded.\\
9943     You-can't-use-the-key~'tabular/bottomrule'~because-you-haven't~
9944     loaded~'booktabs'.\\
9945     This-key-will-be-ignored.

```

```

9946 }
9947 \@@_msg_new:nn { enumitem~not~loaded }
9948 {
9949   enumitem~not~loaded.\\
9950   You~can't~use~the~command~\token_to_str:N\tabularnote\
9951   ~because~you~haven't~loaded~'enumitem'.\\
9952   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9953   ignored~in~the~document.
9954 }
9955 \@@_msg_new:nn { tikz~without~tikz }
9956 {
9957   Tikz~not~loaded.\\
9958   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9959   loaded.~If~you~go~on,~that~key~will~be~ignored.
9960 }
9961 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9962 {
9963   Tikz~not~loaded.\\
9964   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9965   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9966   You~can~go~on~but~you~will~have~another~error~if~you~actually~
9967   use~that~custom~line.
9968 }
9969 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9970 {
9971   Tikz~not~loaded.\\
9972   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9973   command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9974   That~key~will~be~ignored.
9975 }
9976 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9977 {
9978   Erroneous~use.\\
9979   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9980   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9981   The~key~'color'~will~be~discarded.
9982 }
9983 \@@_msg_new:nn { Wrong~last~row }
9984 {
9985   Wrong~number.\\
9986   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9987   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9988   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9989   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9990   without~value~(more~compilations~might~be~necessary).
9991 }
9992 \@@_msg_new:nn { Yet~in~env }
9993 {
9994   Nested~environments.\\
9995   Environments~of~nicematrix~can't~be~nested.\\
9996   This~error~is~fatal.
9997 }
9998 \@@_msg_new:nn { Outside~math~mode }
9999 {
10000   Outside~math~mode.\\
10001   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
10002   (and~not~in~\token_to_str:N \vcenter).\\
10003   This~error~is~fatal.
10004 }
10005 \@@_msg_new:nn { One~letter~allowed }

```

```

10006 {
10007   Bad-name.\
10008   The-value-of-key~'\l_keys_key_str'~must~be~of~length~1.\
10009   It~will~be~ignored.
10010 }
10011 \@@_msg_new:nn { TabularNote~in~CodeAfter }
10012 {
10013   Environment~{TabularNote}~forbidden.\
10014   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
10015   but~*before*~the~\token_to_str:N \CodeAfter.\
10016   This~environment~{TabularNote}~will~be~ignored.
10017 }
10018 \@@_msg_new:nn { varwidth~not~loaded }
10019 {
10020   varwidth~not~loaded.\
10021   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
10022   loaded.\
10023   Your~column~will~behave~like~'p'.
10024 }
10025 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
10026 {
10027   Unknow~key.\
10028   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\
10029   \c_@@_available_keys_str
10030 }
10031 {
10032   The~available~keys~are~(in~alphabetic~order):~
10033   color,~
10034   dotted,~
10035   multiplicity,~
10036   sep-color,~
10037   tikz,~and~total-width.
10038 }
10039
10040 \@@_msg_new:nnn { Unknown~key~for~Block }
10041 {
10042   Unknown~key.\
10043   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10044   \Block.\ It~will~be~ignored. \
10045   \c_@@_available_keys_str
10046 }
10047 {
10048   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10049   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10050   opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10051   and~vlines.
10052 }
10053 \@@_msg_new:nnn { Unknown~key~for~Brace }
10054 {
10055   Unknown~key.\
10056   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10057   \UnderBrace\ and~\token_to_str:N \OverBrace.\
10058   It~will~be~ignored. \
10059   \c_@@_available_keys_str
10060 }
10061 {
10062   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10063   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10064   right-shorten)~and~yshift.
10065 }
10066 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }

```



```

10067 {
10068     Unknown~key.\\
10069     The~key~'\l_keys_key_str'~is~unknown.\\
10070     It~will~be~ignored. \\
10071     \c_@@_available_keys_str
10072 }
10073 {
10074     The~available~keys~are~(in~alphabetic~order):~
10075     delimiters/color,~
10076     rules~(with~the~subkeys~'color'~and~'width'),~
10077     sub-matrix~(several~subkeys)~
10078     and~xdots~(several~subkeys).~
10079     The~latter~is~for~the~command~\token_to_str:N \line.
10080 }
10081 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10082 {
10083     Unknown~key.\\
10084     The~key~'\l_keys_key_str'~is~unknown.\\
10085     It~will~be~ignored. \\
10086     \c_@@_available_keys_str
10087 }
10088 {
10089     The~available~keys~are~(in~alphabetic~order):~
10090     create-cell-nodes,~
10091     delimiters/color~and~
10092     sub-matrix~(several~subkeys).
10093 }
10094 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10095 {
10096     Unknown~key.\\
10097     The~key~'\l_keys_key_str'~is~unknown.\\
10098     That~key~will~be~ignored. \\
10099     \c_@@_available_keys_str
10100 }
10101 {
10102     The~available~keys~are~(in~alphabetic~order):~
10103     'delimiters/color',~
10104     'extra-height',~
10105     'hlines',~
10106     'hvlines',~
10107     'left-xshift',~
10108     'name',~
10109     'right-xshift',~
10110     'rules'~(with~the~subkeys~'color'~and~'width'),~
10111     'slim',~
10112     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10113     and~'right-xshift').\\
10114 }
10115 \@@_msg_new:nnn { Unknown~key~for~notes }
10116 {
10117     Unknown~key.\\
10118     The~key~'\l_keys_key_str'~is~unknown.\\
10119     That~key~will~be~ignored. \\
10120     \c_@@_available_keys_str
10121 }
10122 {
10123     The~available~keys~are~(in~alphabetic~order):~
10124     bottomrule,~
10125     code-after,~
10126     code-before,~
10127     detect-duplicates,~
10128     enumitem-keys,~
10129     enumitem-keys-para,~

```

```

10130     para,~
10131     label-in-list,~
10132     label-in-tabular~and~
10133     style.
10134 }
10135 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10136 {
10137     Unknown~key.\\
10138     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10139     \token_to_str:N \RowStyle. \\
10140     That~key~will~be~ignored. \\
10141     \c_@@_available_keys_str
10142 }
10143 {
10144     The~available~keys~are~(in~alphabetic~order):~
10145     bold,~
10146     cell-space-top-limit,~
10147     cell-space-bottom-limit,~
10148     cell-space-limits,~
10149     color,~
10150     fill~(alias:~rowcolor),~
10151     nb-rows,
10152     opacity~and~
10153     rounded-corners.
10154 }
10155 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10156 {
10157     Unknown~key.\\
10158     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10159     \token_to_str:N \NiceMatrixOptions. \\
10160     That~key~will~be~ignored. \\
10161     \c_@@_available_keys_str
10162 }
10163 {
10164     The~available~keys~are~(in~alphabetic~order):~
10165     &-in-blocks,~
10166     allow-duplicate-names,~
10167     ampersand-in-blocks,~
10168     caption-above,~
10169     cell-space-bottom-limit,~
10170     cell-space-limits,~
10171     cell-space-top-limit,~
10172     code-for-first-col,~
10173     code-for-first-row,~
10174     code-for-last-col,~
10175     code-for-last-row,~
10176     corners,~
10177     custom-key,~
10178     create-extra-nodes,~
10179     create-medium-nodes,~
10180     create-large-nodes,~
10181     custom-line,~
10182     delimiters~(several~subkeys),~
10183     end-of-row,~
10184     first-col,~
10185     first-row,~
10186     hlines,~
10187     hvlines,~
10188     hvlines-except-borders,~
10189     last-col,~
10190     last-row,~
10191     left-margin,~
10192     light-syntax,~

```

```

10193 light-syntax-expanded,~
10194 matrix/columns-type,~
10195 no-cell-nodes,~
10196 notes~(several~subkeys),~
10197 nullify-dots,~
10198 pgf-node-code,~
10199 renew-dots,~
10200 renew-matrix,~
10201 respect-arraystretch,~
10202 rounded-corners,~
10203 right-margin,~
10204 rules~(with~the~subkeys~'color'~and~'width'),~
10205 small,~
10206 sub-matrix~(several~subkeys),~
10207 vlines,~
10208 xdots~(several~subkeys).
10209 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10210 \@@_msg_new:nnn { Unknown-key-for-NiceArray }
10211 {
10212   Unknown-key.\
10213   The-key~'\l_keys_key_str'~is~unknown~for~the~environment~
10214   \{NiceArray\}. \
10215   That~key~will~be~ignored. \
10216   \c_@@_available_keys_str
10217 }
10218 {
10219   The-available-keys-are~(in~alphabetic~order):~
10220   &-in-blocks,~
10221   ampersand-in-blocks,~
10222   b,~
10223   baseline,~
10224   c,~
10225   cell-space-bottom-limit,~
10226   cell-space-limits,~
10227   cell-space-top-limit,~
10228   code-after,~
10229   code-for-first-col,~
10230   code-for-first-row,~
10231   code-for-last-col,~
10232   code-for-last-row,~
10233   columns-width,~
10234   corners,~
10235   create-extra-nodes,~
10236   create-medium-nodes,~
10237   create-large-nodes,~
10238   extra-left-margin,~
10239   extra-right-margin,~
10240   first-col,~
10241   first-row,~
10242   hlines,~
10243   hvlines,~
10244   hvlines-except-borders,~
10245   last-col,~
10246   last-row,~
10247   left-margin,~
10248   light-syntax,~
10249   light-syntax-expanded,~
10250   name,~
10251   no-cell-nodes,~
10252   nullify-dots,~
10253   pgf-node-code,~

```

```

10254     renew-dots,~
10255     respect-arraystretch,~
10256     right-margin,~
10257     rounded-corners,~
10258     rules~(with~the~subkeys~'color'~and~'width'),~
10259     small,~
10260     t,~
10261     vlines,~
10262     xdots/color,~
10263     xdots/shorten-start,~
10264     xdots/shorten-end,~
10265     xdots/shorten-and~
10266     xdots/line-style.
10267 }

```

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`).

```

10268 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10269 {
10270   Unknown~key.\\
10271   The~key~'\l_keys_key_str'~is~unknown~for~the~
10272   \@@_full_name_env:. \\
10273   That~key~will~be~ignored. \\
10274   \c_@@_available_keys_str
10275 }
10276 {
10277   The~available~keys~are~(in~alphabetic~order):~
10278   &-in-blocks,~
10279   ampersand-in-blocks,~
10280   b,~
10281   baseline,~
10282   c,~
10283   cell-space-bottom-limit,~
10284   cell-space-limits,~
10285   cell-space-top-limit,~
10286   code-after,~
10287   code-for-first-col,~
10288   code-for-first-row,~
10289   code-for-last-col,~
10290   code-for-last-row,~
10291   columns-type,~
10292   columns-width,~
10293   corners,~
10294   create-extra-nodes,~
10295   create-medium-nodes,~
10296   create-large-nodes,~
10297   extra-left-margin,~
10298   extra-right-margin,~
10299   first-col,~
10300   first-row,~
10301   hlines,~
10302   hvlines,~
10303   hvlines-except-borders,~
10304   l,~
10305   last-col,~
10306   last-row,~
10307   left-margin,~
10308   light-syntax,~
10309   light-syntax-expanded,~
10310   name,~
10311   no-cell-nodes,~
10312   nullify-dots,~
10313   pgf-node-code,~
10314   r,~

```

```

10315 renew-dots,~
10316 respect-arraystretch,~
10317 right-margin,~
10318 rounded-corners,~
10319 rules~(with~the~subkeys~'color'~and~'width'),~
10320 small,~
10321 t,~
10322 vlines,~
10323 xdots/color,~
10324 xdots/shorten-start,~
10325 xdots/shorten-end,~
10326 xdots/shorten-and~
10327 xdots/line-style.
10328 }
10329 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10330 {
10331   Unknown~key.\\
10332   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10333   \{NiceTabular\}. \\
10334   That~key~will~be~ignored. \\
10335   \c_@@_available_keys_str
10336 }
10337 {
10338   The~available~keys~are~(in~alphabetic~order):~
10339   &-in-blocks,~
10340   ampersand-in-blocks,~
10341   b,~
10342   baseline,~
10343   c,~
10344   caption,~
10345   cell-space-bottom-limit,~
10346   cell-space-limits,~
10347   cell-space-top-limit,~
10348   code-after,~
10349   code-for-first-col,~
10350   code-for-first-row,~
10351   code-for-last-col,~
10352   code-for-last-row,~
10353   columns-width,~
10354   corners,~
10355   custom-line,~
10356   create-extra-nodes,~
10357   create-medium-nodes,~
10358   create-large-nodes,~
10359   extra-left-margin,~
10360   extra-right-margin,~
10361   first-col,~
10362   first-row,~
10363   hlines,~
10364   hvlines,~
10365   hvlines-except-borders,~
10366   label,~
10367   last-col,~
10368   last-row,~
10369   left-margin,~
10370   light-syntax,~
10371   light-syntax-expanded,~
10372   name,~
10373   no-cell-nodes,~
10374   notes~(several~subkeys),~
10375   nullify-dots,~
10376   pgf-node-code,~
10377   renew-dots,~

```

```

10378     respect-arraystretch,~
10379     right-margin,~
10380     rounded-corners,~
10381     rules~(with~the~subkeys~'color'~and~'width'),~
10382     short-caption,~
10383     t,~
10384     tabularnote,~
10385     vl原因,~
10386     xdots/color,~
10387     xdots/shorten-start,~
10388     xdots/shorten-end,~
10389     xdots/shorten-and~
10390     xdots/line-style.
10391 }

10392 \@@_msg_new:nnn { Duplicate~name }
10393 {
10394     Duplicate~name.\\
10395     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10396     the~same~environment~name~twice.~You~can~go~on,~but,~
10397     maybe,~you~will~have~incorrect~results~especially~
10398     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10399     message~again,~use~the~key~'allow-duplicate-names'~in~
10400     '\token_to_str:N \NiceMatrixOptions'.\\
10401     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10402     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10403 }
10404 {
10405     The~names~already~defined~in~this~document~are:~
10406     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10407 }

10408 \@@_msg_new:nn { Option~auto~for~columns-width }
10409 {
10410     Erroneous~use.\\
10411     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10412     That~key~will~be~ignored.
10413 }

10414 \@@_msg_new:nn { NiceTabularX~without~X }
10415 {
10416     NiceTabularX~without~X.\\
10417     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10418     However,~you~can~go~on.
10419 }

10420 \@@_msg_new:nn { Preamble~forgotten }
10421 {
10422     Preamble~forgotten.\\
10423     You~have~probably~forgotten~the~preamble~of~your~
10424     \@@_full_name_env:. \\
10425     This~error~is~fatal.
10426 }

10427 \@@_msg_new:nn { Invalid~col~number }
10428 {
10429     Invalid~column~number.\\
10430     A~color~instruction~the~\token_to_str:N \CodeBefore\
10431     specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
10432 }

10433 \@@_msg_new:nn { Invalid~row~number }
10434 {
10435     Invalid~row~number.\\
10436     A~color~instruction~the~\token_to_str:N \CodeBefore\
10437     specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
10438 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	19
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	74
13	The environment <code>{NiceMatrix}</code> and its variants	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	120
19	The command <code>\line</code> accessible in code-after	126
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	130
22	The vertical and horizontal rules	143
23	The empty corners	159
24	The environment <code>{NiceMatrixBlock}</code>	161
25	The extra nodes	163
26	The blocks	167
27	How to draw the dotted lines transparently	191
28	Automatic arrays	192
29	The redefinition of the command <code>\dotfill</code>	193
30	The command <code>\diagbox</code>	193

31	The keyword <code>\CodeAfter</code>	195
32	The delimiters in the preamble	196
33	The command <code>\SubMatrix</code>	197
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	205
35	The command <code>TikzEveryCell</code>	208
36	The command <code>\ShowCellNames</code>	210
37	We process the options at package loading	211
38	About the package underscore	213
39	Error messages of the package	213