

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

December 3, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 7.0x of `nicematrix`, at the date of 2024/11/27.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

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

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

```

2 Collecting options

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

Exemple :

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

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

```

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

```

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

```

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

```

3 Technical definitions

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

```

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

```

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

```

97 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

```

208     \skip_horizontal:N \c_zero_dim
209   }

```

¹See question 99041 on TeX StackExchange.

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

```

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

```

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

```

214 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

4 Parameters

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

```

278 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

```

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

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Idem for the mono-row blocks.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
331 {
```

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

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

```
334   \fi:
```

```
335 }
```

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

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

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

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

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

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

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

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

```

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

342 \bool_new:N \l_@@_bold_row_style_bool

```

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

```

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

```

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

```

349 \tl_new:N \l_@@_code_tl

```

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

```

350 \tl_new:N \l_@@_pgf_node_code_tl

```

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

```

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

```

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

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

```

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

```

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

```

355 \bool_new:N \l_@@_in_code_after_bool

```

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

```

356 \bool_new:N \l_@@_ampersand_bool

```

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

```

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

```

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

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

```

359 \seq_new:N \l_@@_custom_line_commands_seq

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
378 \dim_new:N \g_@@_dp_row_zero_dim
```

```
379 \dim_new:N \g_@@_ht_row_zero_dim
```

```
380 \dim_new:N \g_@@_ht_row_one_dim
```

```
381 \dim_new:N \g_@@_dp_ante_last_row_dim
```

```
382 \dim_new:N \g_@@_ht_last_row_dim
```

```
383 \dim_new:N \g_@@_dp_last_row_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The following counters will be used when drawing the rules.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variables for the exterior rows and columns

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

• First row

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

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

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

• First column

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

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

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


- **Last row**

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

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

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

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

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

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

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

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

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

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

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

Some utilities

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

The following internal parameters are for:

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

```

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

```

5 The command `\tabularnote`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

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

```

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

```

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

```

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

```

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

```

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

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

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

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

```

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

```

```

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

```

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

```

587     \hbox_set:Nn \l_tmpa_box
588     {

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

6 Command for creation of rectangle nodes

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

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

```

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

```

```

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

```

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

```

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

```

7 The options

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

```

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

```

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

```

696 \bool_new:N \l_@@_caption_above_bool

```

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

```

697 \bool_new:N \l_@@_color_inside_bool

```


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To 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.

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

934     name .value_required:n = true ,
935     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
936     code-after .value_required:n = true ,
937     color-inside .code:n =
938         \bool_set_true:N \l_@@_color_inside_bool
939         \bool_set_true:N \l_@@_code_before_bool ,
940     color-inside .value_forbidden:n = true ,
941     colortbl-like .meta:n = color-inside
942 }

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

981 \keys_define:nn { nicematrix / delimiters }
982 {
983     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
984     max-width .default:n = true ,
985     color .tl_set:N = \l_@@_delimiters_color_tl ,
986     color .value_required:n = true ,
987 }

```

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

```

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

```

```

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

```

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

```

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

```

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

```

1046     renew-matrix .code:n = \@@_renew_matrix: ,
1047     renew-matrix .value_forbidden:n = true ,

```


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

```
1048     exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,
```

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

```
1049     columns-width .code:n =
```

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

```
1050     \str_if_eq:eeTF { #1 } { auto }
1051     { \@@_error:n { Option~auto~for~columns~width } }
1052     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
```

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

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

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

```
1066 \NewDocumentCommand \NiceMatrixOptions { m }
1067 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8 Important code used by `{NiceArrayWithDelims}`

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

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

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

```
1154   \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1156   \int_gincr:N \c@jCol
```

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

```
1157   \int_compare:nNnT \c@jCol = \c_one_int
1158     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }
```

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

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

The following command is nullified in the tabulars.

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

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

```

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

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

We will use a version a little more efficient.

```

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

```

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

```

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

```

The following commands are nullified in the tabulars.

```

1182 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1183 {
1184   \m@th % added 2024/11/21
1185   \c_math_toggle_token

```

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

```

1186   \@@_tuning_key_small:
1187 }
1188 \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.

```

1189 \cs_new_protected:Npn \@@_begin_of_row:
1190 {
1191   \int_gincr:N \c@iRow

```

```

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

```

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.

```

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

```

```

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

```

The following command is nullified in the tabulars.

```

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

```

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

```

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

```

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

```

1288     \@@_update_max_cell_width:

```

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

```

1289     \@@_update_for_first_and_last_row:

```

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

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

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

```

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

```

1406 \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)`.

```

1407 [ \str_if_eq:eeTF \l_@@_baseline_tl c c t ]
1408 }

```

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

```

1409 \bool_if:nTF

```

```

1410 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1411 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1412 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1413 \cs_new_protected:Npn \@@_create_row_node:
1414 {
1415   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1416   {
1417     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1418     \@@_create_row_node_i:
1419   }
1420 }
1421 \cs_new_protected:Npn \@@_create_row_node_i:
1422 {

```

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

```

1423   \hbox
1424   {
1425     \bool_if:NT \l_@@_code_before_bool
1426     {
1427       \vtop
1428       {
1429         \skip_vertical:N 0.5\arrayrulewidth
1430         \pgfsys@markposition
1431         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1432         \skip_vertical:N -0.5\arrayrulewidth
1433       }
1434     }
1435     \pgfpicture
1436     \pgfrememberpicturepositiononpagetrue
1437     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1438     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1439     \str_if_empty:NF \l_@@_name_str
1440     {
1441       \pgfnodealias
1442       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1443       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1444     }
1445     \endpgfpicture
1446   }
1447 }

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

We will issue an error only during the first run.

```

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

```

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

```

1496 \hook_gput_code:nnn { begindocument } { . }
1497 {
1498     \IfPackageLoadedTF { colortbl }
1499     {
1500         \cs_set_protected:Npn \@@_everycr:
1501         { \CTeverycr { \noalign { \@@_in_everycr: } } }
1502     }
1503     {
1504         \cs_new_protected:Npn \@@_everycr:
1505         { \everycr { \noalign { \@@_in_everycr: } } }
1506     }
1507 }

```

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

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

```

1508 \hook_gput_code:nnn { begindocument } { . }
1509 {
1510   \IfPackageLoadedTF { booktabs }
1511   {
1512     \cs_new_protected:Npn \@@_patch_booktabs:
1513     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1514   }
1515   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1516 }

```

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`.

```

1517 \cs_new_protected:Npn \@@_some_initialization:
1518 {
1519   \@@_everycr:
1520   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1521   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1522   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1523   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1524   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1525   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1526 }

```

```

1527 \cs_new_protected:Npn \@@_pre_array_ii:
1528 {

```

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

```

1529   \int_gzero:N \g_@@_total_X_weight_int
1530   \@@_expand_clist:N \l_@@_hlines_clist
1531   \@@_expand_clist:N \l_@@_vlines_clist
1532   \@@_patch_booktabs:
1533   \box_clear_new:N \l_@@_cell_box
1534   \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}`).

```

1535   \bool_if:NT \l_@@_small_bool
1536   {
1537     \cs_set_nopar:Npn \arraystretch { 0.47 }
1538     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1539     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1540   }

```

```

1541   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1542   {
1543     \tl_put_right:Nn \@@_begin_of_row:
1544     {
1545       \pgfsys@markposition

```

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

```

1546         { \c_@@_env: - row - \int_use:N \c@iRow - base }
1547     }
1548 }

```

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`.

```

1549 \bool_if:nTF
1550 { \c_@@_recent_array_bool && ! \c_@@_revtex_bool }
1551 {
1552     \cs_set_nopar:Npn \ar@ialign
1553     {
1554         \bool_if:NT \c_@@_testphase_table_bool
1555         \tbl_init_cell_data_for_table:
1556         \@@_some_initialization:
1557         \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`.

```

1558         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1559         \halign
1560     }
1561 }

```

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`.

```

1562 {
1563     \cs_set_nopar:Npn \ialign
1564     {
1565         \@@_some_initialization:
1566         \dim_zero:N \tabskip
1567         \cs_set_eq:NN \ialign \@@_old_ialign:
1568         \halign
1569     }
1570 }

```

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).

```

1571 \bool_if:NT \c_@@_revtex_bool
1572 {
1573     \IfPackageLoadedT { colortbl }
1574     { \cs_set_protected:Npn \CT@setup { } }
1575 }

```

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).

```

1576 \cs_set_eq:NN \@@_old_ldots \ldots
1577 \cs_set_eq:NN \@@_old_cdots \cdots
1578 \cs_set_eq:NN \@@_old_vdots \vdots
1579 \cs_set_eq:NN \@@_old_ddots \ddots
1580 \cs_set_eq:NN \@@_old_iddots \iddots
1581 \bool_if:NTF \l_@@_standard_cline_bool
1582 { \cs_set_eq:NN \cline \@@_standard_cline }
1583 { \cs_set_eq:NN \cline \@@_cline }
1584 \cs_set_eq:NN \Ldots \@@_Ldots
1585 \cs_set_eq:NN \Cdots \@@_Cdots
1586 \cs_set_eq:NN \Vdots \@@_Vdots
1587 \cs_set_eq:NN \Ddots \@@_Ddots

```

```

1588 \cs_set_eq:NN \Iddots \@@_Iddots
1589 \cs_set_eq:NN \Hline \@@_Hline:
1590 \cs_set_eq:NN \Hspace \@@_Hspace:
1591 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1592 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1593 \cs_set_eq:NN \Block \@@_Block:
1594 \cs_set_eq:NN \rotate \@@_rotate:
1595 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1596 \cs_set_eq:NN \dotfill \@@_dotfill:
1597 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1598 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1599 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1600 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1601 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1602   { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1603 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1604 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1605 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1606 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1607 \int_compare:nNtT \l_@@_first_row_int > \c_zero_int
1608   { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1609 \int_compare:nNtT \l_@@_last_row_int < \c_zero_int
1610   { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1611 \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:`.

```

1612 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1613 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1614   { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1615 \@@_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!).

```

1616 \tl_if_exist:NT \l_@@_note_in_caption_tl
1617   {
1618     \tl_if_empty:NF \l_@@_note_in_caption_tl
1619     {
1620       \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1621       \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1622     }
1623   }

```

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).

```

1624 \seq_gclear:N \g_@@_multicolumn_cells_seq
1625 \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).

```

1626 \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`).

```

1627 \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.

```

1628     \int_gzero_new:N \g_@@_col_total_int
1629     \cs_set_eq:NN \@@_ifnextchar \new@ifnextchar
1630     \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.

```

1631     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1632     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1633     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1634     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1635     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1636     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1637     \tl_gclear:N \g_nicematrix_code_before_tl
1638     \tl_gclear:N \g_@@_pre_code_before_tl
1639 }

```

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

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

```

1640 \cs_new_protected:Npn \@@_pre_array:
1641 {
1642     \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1643     \int_gzero_new:N \c@iRow
1644     \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1645     \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).

```

1646     \int_compare:nNnT \l_@@_last_row_int = { -1 }
1647     {
1648         \bool_set_true:N \l_@@_last_row_without_value_bool
1649         \bool_if:NT \g_@@_aux_found_bool
1650         { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1651     }
1652     \int_compare:nNnT \l_@@_last_col_int = { -1 }
1653     {
1654         \bool_if:NT \g_@@_aux_found_bool
1655         { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1656     }

```

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”.

```

1657     \int_compare:nNnT \l_@@_last_row_int > { -2 }
1658     {
1659         \tl_put_right:Nn \@@_update_for_first_and_last_row:
1660         {
1661             \dim_compare:nNnT \g_@@_ht_last_row_dim < { \box_ht:N \l_@@_cell_box }
1662             { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1663             \dim_compare:nNnT \g_@@_dp_last_row_dim < { \box_dp:N \l_@@_cell_box }
1664             { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1665         }
1666     }

```

```

1667 \seq_gclear:N \g_@@_cols_vlism_seq
1668 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1669 \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.

```

1670 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1671 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1672 \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.

```

1673 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value -2 is important.

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

```

1674 \@@_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.

```

1675 \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]`).

```

1676 \dim_zero_new:N \l_@@_left_delim_dim
1677 \dim_zero_new:N \l_@@_right_delim_dim
1678 \bool_if:NTF \g_@@_delims_bool
1679 {

```

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

```

1680 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1681 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1682 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1683 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1684 }
1685 {
1686 \dim_gset:Nn \l_@@_left_delim_dim
1687 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1688 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1689 }

```

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).

```

1690 \hbox_set:Nw \l_@@_the_array_box
1691 \skip_horizontal:N \l_@@_left_margin_dim
1692 \skip_horizontal:N \l_@@_extra_left_margin_dim
1693 \bool_if:NT \c_@@_recent_array_bool
1694 { \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).

```

1695 \m@th
1696 \c_math_toggle_token
1697 \bool_if:NTF \l_@@_light_syntax_bool
1698   { \use:c { @@-light-syntax } }
1699   { \use:c { @@-normal-syntax } }
1700 }

```

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

```

1701 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1702 {
1703   \tl_set:Nn \l_tmpa_tl { #1 }
1704   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1705     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1706   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1707   \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...

```

1708 \@@_pre_array:
1709 }

```

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).

```

1710 \cs_new_protected:Npn \@@_pre_code_before:
1711 {

```

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).

```

1712 \int_set:Nn \c_iRow { \seq_item:Nn \g_@@_size_seq 2 }
1713 \int_set:Nn \c_jCol { \seq_item:Nn \g_@@_size_seq 5 }
1714 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1715 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }

```

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

```

1716 \pgfsys@markposition { \@@_env: - position }
1717 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1718 \pgfpicture
1719 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1720 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1721 {
1722   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1723   \pgfcoordinate { \@@_env: - row - ##1 }
1724   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1725 }

```

Now, the recreation of the col nodes.

```

1726 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1727 {
1728   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1729   \pgfcoordinate { \@@_env: - col - ##1 }
1730   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1731 }

```

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

```

1732 \@@_create_diag_nodes:

```

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

```

1733 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1734 \endpgfpicture

```

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

```

1735 \@@_create_blocks_nodes:
1736 \IfPackageLoadedT { tikz }
1737 {
1738   \tikzset
1739   {
1740     every-picture / .style =
1741     { overlay , name~prefix = \@@_env: - }
1742   }
1743 }
1744 \cs_set_eq:NN \cellcolor \@@_cellcolor
1745 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1746 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1747 \cs_set_eq:NN \rowcolor \@@_rowcolor
1748 \cs_set_eq:NN \rowcolors \@@_rowcolors
1749 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1750 \cs_set_eq:NN \arraycolor \@@_arraycolor
1751 \cs_set_eq:NN \columncolor \@@_columncolor
1752 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1753 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1754 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1755 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1756 }

1757 \cs_new_protected:Npn \@@_exec_code_before:
1758 {

```

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 detect whether we actually have coloring instructions to execute...

```

1759 \clist_map_inline:Nn \l_@@_corners_cells_clist
1760 { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
1761 \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`.

```

1762 \@@_add_to_colors_seq:nn { { nocolor } } { }
1763 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1764 \group_begin:

```

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

```

1765 \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`).

```
1766 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1767 { \@@_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`.

```
1768 \exp_last_unbraced:No \@@_CodeBefore_keys:
1769 \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.

```
1770 \@@_actually_color:
1771 \l_@@_code_before_tl
1772 \q_stop
1773 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1774 \group_end:
1775 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1776 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1777 }
```

```
1778 \keys_define:nn { nicematrix / CodeBefore }
1779 {
1780   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1781   create-cell-nodes .default:n = true ,
1782   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1783   sub-matrix .value_required:n = true ,
1784   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1785   delimiters / color .value_required:n = true ,
1786   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1787 }
1788 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1789 {
1790   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1791   \@@_CodeBefore:w
1792 }
```

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.

```
1793 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1794 {
1795   \bool_if:NT \g_@@_aux_found_bool
1796   {
1797     \@@_pre_code_before:
1798     #1
1799   }
1800 }
```

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.

```
1801 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1802 {
1803   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
```

```

1804 {
1805   \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1806   \pgfcoordinate { \@@_env: - row - ##1 - base }
1807   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1808   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1809   {
1810     \cs_if_exist:cT
1811     { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1812     {
1813       \pgfsys@getposition
1814       { \@@_env: - ##1 - #####1 - NW }
1815       \@@_node_position:
1816       \pgfsys@getposition
1817       { \@@_env: - ##1 - #####1 - SE }
1818       \@@_node_position_i:
1819       \@@_pgf_rect_node:nnn
1820       { \@@_env: - ##1 - #####1 }
1821       { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1822       { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1823     }
1824   }
1825 }
1826 \int_step_inline:nn \c@iRow
1827 {
1828   \pgfnodealias
1829   { \@@_env: - ##1 - last }
1830   { \@@_env: - ##1 - \int_use:N \c@jCol }
1831 }
1832 \int_step_inline:nn \c@jCol
1833 {
1834   \pgfnodealias
1835   { \@@_env: - last - ##1 }
1836   { \@@_env: - \int_use:N \c@iRow - ##1 }
1837 }
1838 \@@_create_extra_nodes:
1839 }

1840 \cs_new_protected:Npn \@@_create_blocks_nodes:
1841 {
1842   \pgfpicture
1843   \pgf@relevantforpicturesizefalse
1844   \pgfrememberpicturepositiononpagetrue
1845   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1846   { \@@_create_one_block_node:nnnnn ##1 }
1847   \endpgfpicture
1848 }

```

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

```

1849 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1850 {
1851   \tl_if_empty:nF { #5 }
1852   {
1853     \@@_qpoint:n { col - #2 }
1854     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1855     \@@_qpoint:n { #1 }
1856     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1857     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1858     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```

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).

```

1859 \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1860 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1861 \@@_pgf_rect_node:nnnnn
1862 { \@@_env: - #5 }
1863 { \dim_use:N \l_tmpa_dim }
1864 { \dim_use:N \l_tmpb_dim }
1865 { \dim_use:N \l_@@_tmpc_dim }
1866 { \dim_use:N \l_@@_tmpd_dim }
1867 }
1868 }

1869 \cs_new_protected:Npn \@@_patch_for_revtext:
1870 {
1871 \cs_set_eq:NN \@addamp \@addamp@LaTeX
1872 \cs_set_eq:NN \@array \@array@array
1873 \cs_set_eq:NN \@tabular \@tabular@array
1874 \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1875 \cs_set_eq:NN \array \array@array
1876 \cs_set_eq:NN \endarray \endarray@array
1877 \cs_set:Npn \endtabular { \endarray $\egroup } % $
1878 \cs_set_eq:NN \@mkpream \@mkpream@array
1879 \cs_set_eq:NN \@classx \@classx@array
1880 \cs_set_eq:NN \insert@column \insert@column@array
1881 \cs_set_eq:NN \@arraycr \@arraycr@array
1882 \cs_set_eq:NN \@xarraycr \@xarraycr@array
1883 \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1884 }

```

10 The environment {NiceArrayWithDelims}

```

1885 \NewDocumentEnvironment { NiceArrayWithDelims }
1886 { m m O { } m ! O { } t \CodeBefore }
1887 {
1888 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1889 \@@_provide_pgfsyspdfmark:
1890 \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.

```

1891 \bgroup

1892 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1893 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1894 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1895 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1896 \int_gzero:N \g_@@_block_box_int
1897 \dim_zero:N \g_@@_width_last_col_dim
1898 \dim_zero:N \g_@@_width_first_col_dim
1899 \bool_gset_false:N \g_@@_row_of_col_done_bool
1900 \str_if_empty:NT \g_@@_name_env_str
1901 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1902 \bool_if:NTF \l_@@_tabular_bool
1903 \mode_leave_vertical:
1904 \@@_test_if_math_mode:
1905 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1906 \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.

```
1907 \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.

```
1908 \cs_if_exist:NT \tikz@library@external@loaded
1909 {
1910     \tikzexternaldisable
1911     \cs_if_exist:NT \ifstandalone
1912     { \tikzset { external / optimize = false } }
1913 }
```

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

```
1914 \int_gincr:N \g_@@_env_int
1915 \bool_if:NF \l_@@_block_auto_columns_width_bool
1916 { \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`).

```
1917 \seq_gclear:N \g_@@_blocks_seq
1918 \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`.

```
1919 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1920 \seq_gclear:N \g_@@_pos_of_xdots_seq
1921 \tl_gclear_new:N \g_@@_code_before_tl
1922 \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.

```
1923 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1924 {
1925     \bool_gset_true:N \g_@@_aux_found_bool
1926     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1927 }
1928 { \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.

```
1929 \tl_gclear:N \g_@@_aux_tl
1930 \tl_if_empty:NF \g_@@_code_before_tl
1931 {
1932     \bool_set_true:N \l_@@_code_before_bool
1933     \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1934 }
1935 \tl_if_empty:NF \g_@@_pre_code_before_tl
1936 { \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`.

```
1937 \bool_if:NTF \g_@@_delims_bool
1938 { \keys_set:nn { nicematrix / pNiceArray } }
1939 { \keys_set:nn { nicematrix / NiceArray } }
1940 { #3 , #5 }
```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```
1941 \@@_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:.

```
1942 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1943 }
```

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

```
1944 {
1945   \bool_if:NTF \l_@@_light_syntax_bool
1946   { \use:c { end @@-light-syntax } }
1947   { \use:c { end @@-normal-syntax } }
1948   \c_math_toggle_token
1949   \skip_horizontal:N \l_@@_right_margin_dim
1950   \skip_horizontal:N \l_@@_extra_right_margin_dim
1951
1952   % awful workaround
1953   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1954   {
1955     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1956     {
1957       \skip_horizontal:N - \l_@@_columns_width_dim
1958       \bool_if:NTF \l_@@_tabular_bool
1959       { \skip_horizontal:n { - 2 \tabcolsep } }
1960       { \skip_horizontal:n { - 2 \arraycolsep } }
1961     }
1962   }
1963   \hbox_set_end:
1964   \bool_if:NT \c_@@_recent_array_bool
1965   { \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.

```
1966 \bool_if:NT \l_@@_width_used_bool
1967 {
1968   \int_if_zero:nT \g_@@_total_X_weight_int
1969   { \@@_error_or_warning:n { width-without-X~columns } }
1970 }
```

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 .

```
1971 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1972 { \@@_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).

```
1973 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1974 {
1975   \bool_if:NF \l_@@_last_row_without_value_bool
1976   {
1977     \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1978     {
1979       \@@_error:n { Wrong-last-row }
1980       \int_gset_eq:NN \l_@@_last_row_int \c_iRow
1981     }
1982   }
1983 }
```

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

```

1984 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1985 \bool_if:NTF \g_@@_last_col_found_bool
1986 { \int_gdecr:N \c@jCol }
1987 {
1988   \int_compare:nNnT \l_@@_last_col_int > { -1 }
1989   { \@@_error:n { last~col~not~used } }
1990 }

```

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

```

1991 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1992 \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).

```

1993 \int_if_zero:nT \l_@@_first_col_int
1994 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

1995 \bool_if:nTF { ! \g_@@_delims_bool }
1996 {
1997   \str_if_eq:eeTF \l_@@_baseline_tl { c }
1998   \@@_use_arraybox_with_notes_c:
1999   {
2000     \str_if_eq:eeTF \l_@@_baseline_tl { b }
2001     \@@_use_arraybox_with_notes_b:
2002     \@@_use_arraybox_with_notes:
2003   }
2004 }

```

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).

```

2005 {
2006   \int_if_zero:nTF \l_@@_first_row_int
2007   {
2008     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2009     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2010   }
2011   { \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

```

2012 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2013 {
2014   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2015   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2016 }
2017 { \dim_zero:N \l_tmpb_dim }
2018 \hbox_set:Nn \l_tmpa_box
2019 {
2020   \m@th % added 2024/11/21
2021   \c_math_toggle_token
2022   \@@_color:o \l_@@_delimiters_color_tl
2023   \exp_after:wN \left \g_@@_left_delim_tl
2024   \vcenter
2025   {

```

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).

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.

```

2026         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2027         \hbox
2028         {
2029             \bool_if:NTF \l_@@_tabular_bool
2030             { \skip_horizontal:N -\tabcolsep }
2031             { \skip_horizontal:N -\arraycolsep }
2032             \@@_use_arraybox_with_notes_c:
2033             \bool_if:NTF \l_@@_tabular_bool
2034             { \skip_horizontal:N -\tabcolsep }
2035             { \skip_horizontal:N -\arraycolsep }
2036         }

```

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

```

2037         \skip_vertical:N -\l_tmpb_dim
2038         \skip_vertical:N \arrayrulewidth
2039     }
2040     \exp_after:wN \right \g_@@_right_delim_tl
2041     \c_math_toggle_token
2042 }

```

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.

```

2043     \bool_if:NTF \l_@@_delimiters_max_width_bool
2044     {
2045         \@@_put_box_in_flow_bis:nn
2046         \g_@@_left_delim_tl
2047         \g_@@_right_delim_tl
2048     }
2049     \@@_put_box_in_flow:
2050 }

```

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).

```

2051     \bool_if:NT \g_@@_last_col_found_bool
2052     { \skip_horizontal:N \g_@@_width_last_col_dim }
2053     \bool_if:NT \l_@@_preamble_bool
2054     {
2055         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2056         { \@@_warning_gredirect_none:n { columns-not-used } }
2057     }
2058     \@@_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.

```

2059     \egroup

```

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

```

2060     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2061     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2062     \iow_now:Ne \@mainaux
2063     {
2064         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2065         { \exp_not:o \g_@@_aux_tl }
2066     }
2067     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2068     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2069 }

```

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 .

```

2070 \cs_new_protected:Npn \@@_compute_width_X:
2071 {
2072   \tl_gput_right:Ne \g_@@_aux_tl
2073   {
2074     \bool_set_true:N \l_@@_X_columns_aux_bool
2075     \dim_set:Nn \l_@@_X_columns_dim
2076     {
2077       \dim_compare:nNnTF
2078       {
2079         \dim_abs:n
2080         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2081       }
2082       <
2083       { 0.001 pt }
2084       { \dim_use:N \l_@@_X_columns_dim }
2085       {
2086         \dim_eval:n
2087         {
2088           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2089           / \int_use:N \g_@@_total_X_weight_int
2090           + \l_@@_X_columns_dim
2091         }
2092       }
2093     }
2094   }
2095 }

```

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.

```

2096 \cs_new_protected:Npn \@@_transform_preamble:
2097 {
2098   \@@_transform_preamble_i:
2099   \@@_transform_preamble_ii:
2100 }
2101 \cs_new_protected:Npn \@@_transform_preamble_i:
2102 {
2103   \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`).

```

2104   \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.

```

2105   \bool_gset_false:N \g_tmpb_bool

```

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

```

2106   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2107   \int_zero:N \l_tmpa_int
2108   \tl_gclear:N \g_@@_array_preamble_tl
2109   \str_if_eq:eeTF \l_@@_vlines_clist { all }

```

```

2110 {
2111   \tl_gset:Nn \g_@@_array_preamble_tl
2112   { ! { \skip_horizontal:N \arrayrulewidth } }
2113 }
2114 {
2115   \clist_if_in:NnT \l_@@_vlines_clist 1
2116   {
2117     \tl_gset:Nn \g_@@_array_preamble_tl
2118     { ! { \skip_horizontal:N \arrayrulewidth } }
2119   }
2120 }

```

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

```

2121   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2122   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2123   \@@_replace_columncolor:
2124 }

```

```

2125 \hook_gput_code:nnn { begindocument } { . }
2126 {
2127   \IfPackageLoadedTF { colortbl }
2128   {

```

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).

```

2129   \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2130   \cs_new_protected:Npn \@@_replace_columncolor:
2131   {
2132     \regex_replace_all:NnN
2133     \c_@@_columncolor_regex
2134     { \c { @@_columncolor_preamble } }
2135     \g_@@_array_preamble_tl
2136   }
2137 }
2138 {
2139   \cs_new_protected:Npn \@@_replace_columncolor:
2140   { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2141 }
2142 }

```

```

2143 \cs_new_protected:Npn \@@_transform_preamble_ii:
2144 {

```

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

```

2145   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2146   {
2147     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2148     { \bool_gset_true:N \g_@@_delims_bool }
2149   }
2150   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2151   \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).

```

2152 \int_if_zero:nTF \l_@@_first_col_int
2153 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2154 {
2155   \bool_if:NF \g_@@_delims_bool
2156   {
2157     \bool_if:NF \l_@@_tabular_bool
2158     {
2159       \clist_if_empty:NT \l_@@_vlines_clist
2160       {
2161         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2162         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2163       }
2164     }
2165   }
2166 }
2167 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2168 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2169 {
2170   \bool_if:NF \g_@@_delims_bool
2171   {
2172     \bool_if:NF \l_@@_tabular_bool
2173     {
2174       \clist_if_empty:NT \l_@@_vlines_clist
2175       {
2176         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2177         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2178       }
2179     }
2180   }
2181 }

```

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`).

```

2182 \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
2183 {

```

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.

```

2184   \bool_if:NF \c_@@_testphase_table_bool
2185   {
2186     \tl_gput_right:Nn \g_@@_array_preamble_tl
2187     { > { \@@_error_too_much_cols: } l }
2188   }
2189 }
2190 }

```

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.

```

2191 \cs_new_protected:Npn \@@_rec_preamble:n #1
2192 {

```

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

```

2193   \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2194   { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2195   {

```

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`.

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

```

2196     \cs_if_exist:cTF { NC @ find @ #1 }
2197     {
2198         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2199         \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2200     }
2201     {
2202         \str_if_eq:nnTF { #1 } { S }
2203         { \@@_fatal:n { unknown~column~type~S } }
2204         { \@@_fatal:nn { unknown~column~type } { #1 } }
2205     }
2206 }
2207 }

```

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

```

2208 \cs_new_protected:Npn \@@_c #1
2209 {
2210     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2211     \tl_gclear:N \g_@@_pre_cell_tl
2212     \tl_gput_right:Nn \g_@@_array_preamble_tl
2213     { > \@@_cell_begin: c < \@@_cell_end: }

```

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

```

2214     \int_gincr:N \c@jCol
2215     \@@_rec_preamble_after_col:n
2216 }

2217 \cs_new_protected:Npn \@@_l #1
2218 {
2219     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2220     \tl_gclear:N \g_@@_pre_cell_tl
2221     \tl_gput_right:Nn \g_@@_array_preamble_tl
2222     {
2223         > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2224         l
2225         < \@@_cell_end:
2226     }
2227     \int_gincr:N \c@jCol
2228     \@@_rec_preamble_after_col:n
2229 }

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

```

For `!` and `@`

```

2243 \cs_new_protected:cpn { @@ _ \token_to_str:N ! } #1 #2
2244 {
2245     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2246     \@@_rec_preamble:n
2247 }
2248 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2249 \cs_new_protected:cpn { @@ _ | } #1
2250 {
\l_tmpa_int is the number of successive occurrences of |
2251 \int_incr:N \l_tmpa_int
2252 \@@_make_preamble_i_i:n
2253 }

2254 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2255 {
2256 \str_if_eq:nnTF { #1 } { | }
2257 { \use:c { @@ _ | } | }
2258 { \@@_make_preamble_i_ii:nn { } #1 }
2259 }

2260 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2261 {
2262 \str_if_eq:nnTF { #2 } { [ ] }
2263 { \@@_make_preamble_i_iii:nw { #1 } [ ] }
2264 { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2265 }
2266 \cs_new_protected:Npn \@@_make_preamble_i_iii:nw #1 [ #2 ]
2267 { \@@_make_preamble_i_iii:nn { #1 , #2 } }

2268 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2269 {
2270 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2271 \tl_gput_right:Ne \g_@@_array_preamble_tl
2272 {

```

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

```

2273 \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
2274 }
2275 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2276 {
2277 \@@_vline:n
2278 {
2279 position = \int_eval:n { \c@jCol + 1 } ,
2280 multiplicity = \int_use:N \l_tmpa_int ,
2281 total-width = \dim_use:N \l_@@_rule_width_dim ,
2282 #2
2283 }

```

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.

```

2284 }
2285 \int_zero:N \l_tmpa_int
2286 \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2287 \@@_rec_preamble:n #1
2288 }

2289 \cs_new_protected:cpn { @@ _ > } #1 #2
2290 {
2291 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2292 \@@_rec_preamble:n
2293 }

2294 \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.

```

2295 \keys_define:nn { nicematrix / p-column }
2296 {
2297 r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,

```

```

2298     r .value_forbidden:n = true ,
2299     c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2300     c .value_forbidden:n = true ,
2301     l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2302     l .value_forbidden:n = true ,
2303     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2304     S .value_forbidden:n = true ,
2305     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2306     p .value_forbidden:n = true ,
2307     t .meta:n = p ,
2308     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2309     m .value_forbidden:n = true ,
2310     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2311     b .value_forbidden:n = true
2312 }

```

For p but also b and m.

```

2313 \cs_new_protected:Npn \@@_p #1
2314 {
2315     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2316     \@@_make_preamble_ii_i:n
2317 }
2318 \cs_set_eq:NN \@@_b \@@_p
2319 \cs_set_eq:NN \@@_m \@@_p
2320 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2321 {
2322     \str_if_eq:nnTF { #1 } { [ ]
2323     { \@@_make_preamble_ii_ii:w [ ]
2324     { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2325     }
2326 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2327 { \@@_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.

```

2328 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2329 {

```

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).

```

2330     \str_set:Nn \l_@@_hpos_col_str { j }
2331     \@@_keys_p_column:n { #1 }
2332     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2333 }
2334 \cs_new_protected:Npn \@@_keys_p_column:n #1
2335 { \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.

```

2336 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2337 {
2338     \use:e
2339     {
2340         \@@_make_preamble_ii_v:nnnnnnnn
2341         { \str_if_eq:eeTF \l_@@_vpos_col_str { p } { t } { b } }
2342         { \dim_eval:n { #1 } }
2343         {

```

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.

```

2344       \str_if_eq:eeTF \l_@@_hpos_col_str { j }
2345       { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2346       {

```

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

```

2347       \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2348       { \str_lowercase:o \l_@@_hpos_col_str }
2349   }
2350   \IfPackageLoadedTF { ragged2e }
2351   {
2352       \str_case:on \l_@@_hpos_col_str
2353       {
2354           c { \exp_not:N \Centering }
2355           l { \exp_not:N \RaggedRight }
2356           r { \exp_not:N \RaggedLeft }
2357       }
2358   }
2359   {
2360       \str_case:on \l_@@_hpos_col_str
2361       {
2362           c { \exp_not:N \centering }
2363           l { \exp_not:N \raggedright }
2364           r { \exp_not:N \raggedleft }
2365       }
2366   }
2367   #3
2368   }
2369   { \str_if_eq:eeT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2370   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2371   { \str_if_eq:eeT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2372   { #2 }
2373   {
2374       \str_case:onF \l_@@_hpos_col_str
2375       {
2376           { j } { c }
2377           { si } { c }
2378       }

```

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

```

2379       { \str_lowercase:o \l_@@_hpos_col_str }
2380   }
2381   }

```

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

```

2382       \int_gincr:N \c@jCol
2383       \@@_rec_preamble_after_col:n
2384   }

```

#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*.

```

2385 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2386 {
2387   \str_if_eq:eeTF \l_@@_hpos_col_str { si }
2388   {
2389     \tl_gput_right:Nn \g_@@_array_preamble_tl
2390     { > \@@_test_if_empty_for_S: }
2391   }
2392   {
2393     \tl_gput_right:Nn \g_@@_array_preamble_tl
2394     { > \@@_test_if_empty: }
2395   }
2396   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2397   \tl_gclear:N \g_@@_pre_cell_tl
2398   \tl_gput_right:Nn \g_@@_array_preamble_tl
2399   {
2400     > {

```

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.

```

2401       \dim_set:Nn \l_@@_col_width_dim { #2 }
2402       \bool_if:NT \c_@@_testphase_table_bool
2403       { \tag_struct_begin:n { tag = Div } }
2404       \@@_cell_begin:

```

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

```

2405       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2406       \everypar
2407       {
2408         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2409         \everypar { }
2410       }
2411       \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.).

```

2412       #3

```

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

```

2413       \g_@@_row_style_tl
2414       \arraybackslash
2415       #5
2416     }
2417     #8
2418     < {
2419       #6

```

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

```

2420       \@finalstrut \@arstrutbox
2421       \use:c { end #7 }

```

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

```

2422       #4
2423       \@@_cell_end:
2424       \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2425     }
2426   }
2427 }

```

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

```

2428 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
2429 {

```

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 `&`).

```

2430 \group_align_safe_begin:
2431 \peek_meaning:NTF &
2432 {
2433   \group_align_safe_end:
2434   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2435   {

```

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

```

2436     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2437     \skip_horizontal:N \l_@@_col_width_dim
2438   }
2439 }
2440 { \group_align_safe_end: }
2441 }
2442 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2443 {
2444   \peek_meaning:NT \__siunitx_table_skip:n
2445   { \bool_gset_true:N \g_@@_empty_cell_bool }
2446 }

```

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

```

2447 \cs_new_protected:Npn \@@_center_cell_box:
2448 {

```

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

```

2449   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2450   {
2451     \int_compare:nNnT
2452     { \box_ht:N \l_@@_cell_box }
2453     >

```

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).

```

2454     { \box_ht:N \strutbox }
2455     {
2456       \hbox_set:Nn \l_@@_cell_box
2457       {
2458         \box_move_down:nn
2459         {
2460           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2461             + \baselineskip ) / 2
2462         }
2463         { \box_use:N \l_@@_cell_box }
2464       }
2465     }
2466   }
2467 }

```

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

```

2468 \cs_new_protected:Npn \@@_V #1 #2
2469 {

```

```

2470 \str_if_eq:nnTF { #1 } { [ ] }
2471 { \@@_make_preamble_V_i:w [ ] }
2472 { \@@_make_preamble_V_i:w [ ] { #2 } }
2473 }
2474 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2475 { \@@_make_preamble_V_ii:nn { #1 } }
2476 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2477 {
2478 \str_set:Nn \l_@@_vpos_col_str { p }
2479 \str_set:Nn \l_@@_hpos_col_str { j }
2480 \@@_keys_p_column:n { #1 }
2481 \IfPackageLoadedTF { varwidth }
2482 { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2483 {
2484 \@@_error_or_warning:n { varwidth-not-loaded }
2485 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2486 }
2487 }

```

For w and W

```

2488 \cs_new_protected:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2489 \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.

```

2490 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2491 {
2492 \str_if_eq:nnTF { #3 } { s }
2493 { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2494 { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2495 }

```

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.

```

2496 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2497 {
2498 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2499 \tl_gclear:N \g_@@_pre_cell_tl
2500 \tl_gput_right:Nn \g_@@_array_preamble_tl
2501 {
2502 > {
2503 \dim_set:Nn \l_@@_col_width_dim { #2 }
2504 \@@_cell_begin:
2505 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2506 }
2507 c
2508 < {
2509 \@@_cell_end_for_w_s:
2510 #1
2511 \@@_adjust_size_box:
2512 \box_use_drop:N \l_@@_cell_box
2513 }
2514 }
2515 \int_gincr:N \c@jCol
2516 \@@_rec_preamble_after_col:n
2517 }

```

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

```

2518 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4

```

```

2519 {
2520   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2521   \tl_gclear:N \g_@@_pre_cell_tl
2522   \tl_gput_right:Nn \g_@@_array_preamble_tl
2523   {
2524     > {

```

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.

```

2525       \dim_set:Nn \l_@@_col_width_dim { #4 }
2526       \hbox_set:Nw \l_@@_cell_box
2527       \@@_cell_begin:
2528       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2529     }
2530     c
2531     < {
2532       \@@_cell_end:
2533       \hbox_set_end:
2534       #1
2535       \@@_adjust_size_box:
2536       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2537     }
2538   }

```

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

```

2539   \int_gincr:N \c@jCol
2540   \@@_rec_preamble_after_col:n
2541 }

```

```

2542 \cs_new_protected:Npn \@@_special_W:
2543 {
2544   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2545   { \@@_warning:n { W~warning } }
2546 }

```

For `S` (of `siunitx`).

```

2547 \cs_new_protected:Npn \@@_S #1 #2
2548 {
2549   \str_if_eq:nnTF { #2 } { [ ] }
2550   { \@@_make_preamble_S:w [ ] }
2551   { \@@_make_preamble_S:w [ ] { #2 } }
2552 }
2553 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2554 { \@@_make_preamble_S:i:n { #1 } }
2555 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2556 {
2557   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2558   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2559   \tl_gclear:N \g_@@_pre_cell_tl
2560   \tl_gput_right:Nn \g_@@_array_preamble_tl
2561   {
2562     > {
2563       \@@_cell_begin:
2564       \keys_set:nn { siunitx } { #1 }
2565       \siunitx_cell_begin:w
2566     }
2567     c
2568     < { \siunitx_cell_end: \@@_cell_end: }
2569   }

```

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

```

2570   \int_gincr:N \c@jCol
2571   \@@_rec_preamble_after_col:n
2572 }

```

For (, [and \{.

```

2573 \cs_new_protected:cpn { @@ _ \token_to_str:N ( } #1 #2
2574 {
2575   \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.

```

2576   \int_if_zero:nTF \c@jCol
2577   {
2578     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2579     {

```

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

```

2580       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2581       \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2582       \@@_rec_preamble:n #2
2583     }
2584   {
2585     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2586     \@@_make_preamble_iv:nn { #1 } { #2 }
2587   }
2588 }
2589 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2590 }
2591 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2592 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2593 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2594 {
2595   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2596   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2597   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2598   {
2599     \@@_error:nn { delimiter~after~opening } { #2 }
2600     \@@_rec_preamble:n
2601   }
2602   { \@@_rec_preamble:n #2 }
2603 }

```

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

```

2604 \cs_new_protected:cpn { @@ _ \token_to_str:N \left } #1
2605 { \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}).

```

2606 \cs_new_protected:cpn { @@ _ \token_to_str:N ) } #1 #2
2607 {
2608   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2609   \tl_if_in:nnTF { ) ] \} } { #2 }
2610   { \@@_make_preamble_v:nnn #1 #2 }
2611   {
2612     \str_if_eq:nnTF { \@@_stop: } { #2 }
2613     {
2614       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2615       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2616       {
2617         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2618         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2619         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2620         \@@_rec_preamble:n #2
2621       }
2622     }

```

```

2623     {
2624       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2625         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2626       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2627         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2628       \@@_rec_preamble:n #2
2629     }
2630   }
2631 }
2632 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2633 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2634 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2635 {
2636   \str_if_eq:nnTF { \@@_stop: } { #3 }
2637   {
2638     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2639     {
2640       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2641       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2642         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2643       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2644     }
2645     {
2646       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2647       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2648         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2649       \@@_error:nn { double~closing~delimiter } { #2 }
2650     }
2651   }
2652   {
2653     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2654       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2655     \@@_error:nn { double~closing~delimiter } { #2 }
2656     \@@_rec_preamble:n #3
2657   }
2658 }
2659 \cs_new_protected:cpn { @@ _ \token_to_str:N \right } #1
2660 { \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 @{...}.

```

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

```

```

2680     }
2681   }
2682   \@@_rec_preamble:n { #1 }
2683 }
2684 }
2685 }
2686 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2687 {
2688   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2689   \@@_rec_preamble_after_col:n
2690 }

```

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.

```

2691 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2692 {
2693   \str_if_eq:eeTF \l_@@_vlines_clist { all }
2694   {
2695     \tl_gput_right:Nn \g_@@_array_preamble_tl
2696     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2697   }
2698   {
2699     \clist_if_in:NneTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2700     {
2701       \tl_gput_right:Nn \g_@@_array_preamble_tl
2702       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2703     }
2704     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2705   }
2706   \@@_rec_preamble:n
2707 }
2708 \cs_new_protected:cpn { @@ _ * } #1 #2 #3
2709 {
2710   \tl_clear:N \l_tmpa_tl
2711   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2712   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2713 }

```

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.

```

2714 \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`.

```

2715 \cs_new_protected:Npn \@@_X #1 #2
2716 {
2717   \str_if_eq:nnTF { #2 } { [ ]
2718     { \@@_make_preamble_X:w [ ] }
2719     { \@@_make_preamble_X:w [ ] #2 }
2720   }
2721 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2722 { \@@_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`).

```

2723 \keys_define:nn { nicematrix / X-column }
2724 { 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.

```
2725 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2726 {
```

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).

```
2727 \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).

```
2728 \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.

```
2729 \int_zero_new:N \l_@@_weight_int
2730 \int_set_eq:NN \l_@@_weight_int \c_one_int
2731 \@@_keys_p_column:n { #1 }
```

The unknown keys are put in \l_tmpa_tl

```
2732 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2733 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2734 {
2735   \@@_error_or_warning:n { negative~weight }
2736   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2737 }
2738 \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).

```
2739 \bool_if:NTF \l_@@_X_columns_aux_bool
2740 {
2741   \@@_make_preamble_ii_iv:nnn
2742   { \l_@@_weight_int \l_@@_X_columns_dim }
2743   { minipage }
2744   { \@@_no_update_width: }
2745 }
2746 {
2747   \tl_gput_right:Nn \g_@@_array_preamble_tl
2748   {
2749     > {
2750       \@@_cell_begin:
2751       \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.

```
2752 \NotEmpty
```

The following code will nullify the box of the cell.

```
2753 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2754 { \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.

```
2755 \begin { minipage } { 5 cm } \arraybackslash
2756 }
2757 c
2758 < {
2759   \end { minipage }
2760   \@@_cell_end:
2761 }
2762 }
```



```

2763         \int_gincr:N \c@jCol
2764         \@@_rec_preamble_after_col:n
2765     }
2766 }

2767 \cs_new_protected:Npn \@@_no_update_width:
2768 {
2769     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2770     { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2771 }

```

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

```

2772 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2773 {
2774     \seq_gput_right:Ne \g_@@_cols_vlism_seq
2775     { \int_eval:n { \c@jCol + 1 } }
2776     \tl_gput_right:Ne \g_@@_array_preamble_tl
2777     { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2778     \@@_rec_preamble:n
2779 }

```

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.

```

2780 \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).

```

2781 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2782 { \@@_fatal:n { Preamble-forgotten } }
2783 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2784 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2785 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }

```

12 The redefinition of `\multicolumn`

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

```

2786 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2787 {

```

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`.

```

2788     \multispan { #1 }
2789     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2790     \begingroup
2791     \bool_if:NT \c_@@_testphase_table_bool
2792     { \tbl_update_multicolumn_cell_data:n { #1 } }
2793     \cs_set_nopar:Npn \@addamp
2794     { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2795     \tl_gclear:N \g_@@_preamble_tl
2796     \@@_make_m_preamble:n #2 \q_stop

```

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

```

2797 \exp_args:No \mkpream \g_@@_preamble_tl
2798 \addtopreamble \empty
2799 \endgroup
2800 \bool_if:NT \c_@@_recent_array_bool
2801 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2802 \int_compare:nNtT { #1 } > \c_one_int
2803 {
2804   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2805   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2806   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2807   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2808   {
2809     {
2810       \int_if_zero:nTF \c@jCol
2811       { \int_eval:n { \c@iRow + 1 } }
2812       { \int_use:N \c@iRow }
2813     }
2814     { \int_eval:n { \c@jCol + 1 } }
2815     {
2816       \int_if_zero:nTF \c@jCol
2817       { \int_eval:n { \c@iRow + 1 } }
2818       { \int_use:N \c@iRow }
2819     }
2820     { \int_eval:n { \c@jCol + #1 } }
2821     { } % for the name of the block
2822   }
2823 }

```

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

```

2824 \RenewDocumentCommand \cellcolor { 0 { } m }
2825 {
2826   \@@_test_color_inside:
2827   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2828   {
2829     \@@_rectanglecolor [ ##1 ]
2830     { \exp_not:n { ##2 } }
2831     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2832     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2833   }
2834   \ignorespaces
2835 }

```

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

```

2836 \cs_set_nopar:Npn \@sharp { #3 }
2837 \@arstrut
2838 \@preamble
2839 \null

```

We add some lines.

```

2840 \int_gadd:Nn \c@jCol { #1 - 1 }
2841 \int_compare:nNtT \c@jCol > \g_@@_col_total_int
2842 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2843 \ignorespaces
2844 }

```

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`.

```

2845 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2846 {
2847   \str_case:nnF { #1 }
2848   {
2849     c { \@@_make_m_preamble_i:n #1 }
2850     l { \@@_make_m_preamble_i:n #1 }
2851     r { \@@_make_m_preamble_i:n #1 }
2852     > { \@@_make_m_preamble_ii:nn #1 }
2853     ! { \@@_make_m_preamble_ii:nn #1 }
2854     @ { \@@_make_m_preamble_ii:nn #1 }
2855     | { \@@_make_m_preamble_iii:n #1 }
2856     p { \@@_make_m_preamble_iv:nnn t #1 }
2857     m { \@@_make_m_preamble_iv:nnn c #1 }
2858     b { \@@_make_m_preamble_iv:nnn b #1 }
2859     w { \@@_make_m_preamble_v:nnnn { } #1 }
2860     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2861     \q_stop { }
2862   }
2863   {
2864     \cs_if_exist:cTF { NC @ find @ #1 }
2865     {
2866       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2867       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2868     }
2869     {
2870       \str_if_eq:nnTF { #1 } { S }
2871       { \@@_fatal:n { unknown~column~type~S } }
2872       { \@@_fatal:nn { unknown~column~type } { #1 } }
2873     }
2874   }
2875 }

```

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

```

2876 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2877 {
2878   \tl_gput_right:Nn \g_@@_preamble_tl
2879   {
2880     > { \@@_cell_begin: \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2881     #1
2882     < \@@_cell_end:
2883   }

```

We test for the presence of a `<`.

```

2884   \@@_make_m_preamble_x:n
2885 }

```

For `>`, `!` and `@`

```

2886 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2887 {
2888   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2889   \@@_make_m_preamble:n
2890 }

```

For `|`

```

2891 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2892 {
2893   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2894   \@@_make_m_preamble:n
2895 }

```

For p, m and b

```

2896 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2897 {
2898   \tl_gput_right:Nn \g_@@_preamble_tl
2899   {
2900     > {
2901       \@@_cell_begin:
2902       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2903       \mode_leave_vertical:
2904       \arraybackslash
2905       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2906     }
2907     c
2908     < {
2909       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2910       \end { minipage }
2911       \@@_cell_end:
2912     }
2913   }

```

We test for the presence of a <.

```

2914   \@@_make_m_preamble_x:n
2915 }

```

For w and W

```

2916 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2917 {
2918   \tl_gput_right:Nn \g_@@_preamble_tl
2919   {
2920     > {
2921       \dim_set:Nn \l_@@_col_width_dim { #4 }
2922       \hbox_set:Nw \l_@@_cell_box
2923       \@@_cell_begin:
2924       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2925     }
2926     c
2927     < {
2928       \@@_cell_end:
2929       \hbox_set_end:
2930       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2931       #1
2932       \@@_adjust_size_box:
2933       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2934     }
2935   }

```

We test for the presence of a <.

```

2936   \@@_make_m_preamble_x:n
2937 }

```

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

```

2938 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2939 {
2940   \str_if_eq:nnTF { #1 } { < }
2941   \@@_make_m_preamble_ix:n
2942   { \@@_make_m_preamble:n { #1 } }
2943 }
2944 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2945 {
2946   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2947   \@@_make_m_preamble_x:n
2948 }

```

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`).

```

2949 \cs_new_protected:Npn \@@_put_box_in_flow:
2950 {
2951   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2952   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2953   \str_if_eq:eeTF \l_@@_baseline_tl { c }
2954   { \box_use_drop:N \l_tmpa_box }
2955   \@@_put_box_in_flow_i:
2956 }

```

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

```

2957 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2958 {
2959   \pgfpicture
2960   \@@_qpoint:n { row - 1 }
2961   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2962   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2963   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2964   \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).

```

2965   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2966   {
2967     \int_set:Nn \l_tmpa_int
2968     {
2969       \str_range:Nnn
2970       \l_@@_baseline_tl
2971       6
2972       { \tl_count:o \l_@@_baseline_tl }
2973     }
2974     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2975   }
2976   {
2977     \str_if_eq:eeTF \l_@@_baseline_tl { t }
2978     { \int_set_eq:NN \l_tmpa_int \c_one_int }
2979     {
2980       \str_if_eq:onTF \l_@@_baseline_tl { b }
2981       { \int_set_eq:NN \l_tmpa_int \c@iRow }
2982       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2983     }
2984     \bool_lazy_or:nnT
2985     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2986     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2987     {
2988       \@@_error:n { bad-value-for-baseline }
2989       \int_set_eq:NN \l_tmpa_int \c_one_int
2990     }
2991     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2992     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2993   }
2994   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2995   \endpgfpicture
2996   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2997   \box_use_drop:N \l_tmpa_box
2998 }

```

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).

```
2999 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3000 {
```

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.

```
3001 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3002 {
3003   \int_compare:nNt \c@jCol > \c_one_int
3004   {
3005     \box_set_wd:Nn \l_@@_the_array_box
3006     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3007   }
3008 }
```

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

```
3009 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3010 \bool_if:NT \l_@@_caption_above_bool
3011 {
3012   \tl_if_empty:NF \l_@@_caption_tl
3013   {
3014     \bool_set_false:N \g_@@_caption_finished_bool
3015     \int_gzero:N \c@tabularnote
3016     \@@_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).

```
3017   \int_compare:nNt \g_@@_notes_caption_int > \c_zero_int
3018   {
3019     \tl_gput_right:Ne \g_@@_aux_tl
3020     {
3021       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3022       { \int_use:N \g_@@_notes_caption_int }
3023     }
3024     \int_gzero:N \g_@@_notes_caption_int
3025   }
3026 }
3027 }
```

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

```
3028 \hbox
3029 {
3030   \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.

```
3031 \@@_create_extra_nodes:
3032 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3033 }
```

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

```
3034 \bool_lazy_any:nT
3035 {
3036   { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

3037     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3038     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3039   }
3040   \@@_insert_tabularnotes:
3041   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3042   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3043   \end { minipage }
3044 }

```

```

3045 \cs_new_protected:Npn \@@_insert_caption:
3046 {
3047   \tl_if_empty:NF \l_@@_caption_tl
3048   {
3049     \cs_if_exist:NTF \@capytype
3050     { \@@_insert_caption_i: }
3051     { \@@_error:n { caption~outside~float } }
3052   }
3053 }

```

```

3054 \cs_new_protected:Npn \@@_insert_caption_i:
3055 {
3056   \group_begin:

```

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

```

3057   \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.

```

3058   \IfPackageLoadedT { floatrow }
3059   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3060   \tl_if_empty:NTF \l_@@_short_caption_tl
3061   { \caption }
3062   { \caption [ \l_@@_short_caption_tl ] }
3063   { \l_@@_caption_tl }

```

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That's why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

```

3064   \bool_if:NF \g_@@_caption_finished_bool
3065   {
3066     \bool_gset_true:N \g_@@_caption_finished_bool
3067     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3068     \int_gzero:N \c@tabularnote
3069   }
3070   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3071   \group_end:
3072 }

```

```

3073 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3074 {
3075   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3076   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3077 }

```

```

3078 \cs_new_protected:Npn \@@_insert_tabularnotes:
3079 {
3080   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3081   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3082   \skip_vertical:N 0.65ex

```

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

```

3083   \group_begin:
3084   \l_@@_notes_code_before_tl
3085   \tl_if_empty:NF \g_@@_tabularnote_tl
3086   {
3087     \g_@@_tabularnote_tl \par
3088     \tl_gclear:N \g_@@_tabularnote_tl
3089   }

```

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.

```

3090   \int_compare:nNnT \c@tabularnote > \c_zero_int
3091   {
3092     \bool_if:NTF \l_@@_notes_para_bool
3093     {
3094       \begin { tabularnotes* }
3095       \seq_map_inline:Nn \g_@@_notes_seq
3096       { \@@_one_tabularnote:nn ##1 }
3097       \strut
3098       \end { tabularnotes* }

```

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

```

3099     \par
3100   }
3101   {
3102     \tabularnotes
3103     \seq_map_inline:Nn \g_@@_notes_seq
3104     { \@@_one_tabularnote:nn ##1 }
3105     \strut
3106     \endtabularnotes
3107   }
3108 }
3109 \unskip
3110 \group_end:
3111 \bool_if:NT \l_@@_notes_bottomrule_bool
3112 {
3113   \IfPackageLoadedTF { booktabs }
3114   {

```

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

```

3115     \skip_vertical:N \aboverulesep

```

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

```

3116     { \CT@arc@ \hrule height \heavyrulewidth }
3117   }
3118   { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3119 }
3120 \l_@@_notes_code_after_tl
3121 \seq_gclear:N \g_@@_notes_seq
3122 \seq_gclear:N \g_@@_notes_in_caption_seq
3123 \int_gzero:N \c@tabularnote
3124 }

```

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.

```

3125 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3126 {
3127   \tl_if_novalue:nTF { #1 }
3128   { \item }
3129   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3130 }

```


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`.

```

3131 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3132 {
3133   \pgfpicture
3134     \@@_qpoint:n { row - 1 }
3135     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3136     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3137     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3138   \endpgfpicture
3139   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3140   \int_if_zero:nT \l_@@_first_row_int
3141   {
3142     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3143     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3144   }
3145   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3146 }

```

Now, the general case.

```

3147 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3148 {

```

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

```

3149   \str_if_eq:eeT \l_@@_baseline_tl { t }
3150   { \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`.

```

3151   \pgfpicture
3152   \@@_qpoint:n { row - 1 }
3153   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3154   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3155   {
3156     \int_set:Nn \l_tmpa_int
3157     {
3158       \str_range:Nnn
3159         \l_@@_baseline_tl
3160         6
3161         { \tl_count:o \l_@@_baseline_tl }
3162     }
3163     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3164   }
3165   {
3166     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3167     \bool_lazy_or:nnT
3168     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3169     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3170     {
3171       \@@_error:n { bad-value-for-baseline }
3172       \int_set:Nn \l_tmpa_int 1
3173     }
3174     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3175   }
3176   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3177   \endpgfpicture
3178   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3179   \int_if_zero:nT \l_@@_first_row_int
3180   {
3181     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3182     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3183   }
3184   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3185 }
```

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.

```
3186 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3187 {
```

We will compute the real width of both delimiters used.

```
3188 \dim_zero_new:N \l_@@_real_left_delim_dim
3189 \dim_zero_new:N \l_@@_real_right_delim_dim
3190 \hbox_set:Nn \l_tmpb_box
3191 {
3192   \m@th % added 2024/11/21
3193   \c_math_toggle_token
3194   \left #1
3195   \vcenter
3196   {
3197     \vbox_to_ht:nn
3198     { \box_ht_plus_dp:N \l_tmpa_box }
3199     { }
3200   }
3201   \right .
3202   \c_math_toggle_token
3203 }
3204 \dim_set:Nn \l_@@_real_left_delim_dim
3205 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3206 \hbox_set:Nn \l_tmpb_box
3207 {
3208   \m@th % added 2024/11/21
3209   \c_math_toggle_token
3210   \left .
3211   \vbox_to_ht:nn
3212   { \box_ht_plus_dp:N \l_tmpa_box }
3213   { }
3214   \right #2
3215   \c_math_toggle_token
3216 }
3217 \dim_set:Nn \l_@@_real_right_delim_dim
3218 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3219 \skip_horizontal:N \l_@@_left_delim_dim
3220 \skip_horizontal:N -\l_@@_real_left_delim_dim
3221 \@@_put_box_in_flow:
3222 \skip_horizontal:N \l_@@_right_delim_dim
3223 \skip_horizontal:N -\l_@@_real_right_delim_dim
3224 }
```

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).

```
3225 \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).

```
3226 {
3227   \peek_remove_spaces:n
3228   {
```

```

3229     \peek_meaning:NTF \end
3230     \@@_analyze_end:Nn
3231     {
3232         \@@_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`).

```

3233         \@@_array:o \g_@@_array_preamble_tl
3234     }
3235 }
3236 }
3237 {
3238     \@@_create_col_nodes:
3239     \endarray
3240 }

```

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

```

3241 \NewDocumentEnvironment { @@-light-syntax } { b }
3242 {

```

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`.

```

3243     \tl_if_empty:nT { #1 }
3244     { \@@_fatal:n { empty-environment } }
3245     \tl_if_in:nnT { #1 } { & }
3246     { \@@_fatal:n { ampersand-in~light-syntax } }
3247     \tl_if_in:nnT { #1 } { \ }
3248     { \@@_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`.

```

3249     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3250 }

```

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.

```

3251 {
3252     \@@_create_col_nodes:
3253     \endarray
3254 }
3255 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3256 {
3257     \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).

```

3258     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3259     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3260     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3261     \seq_set_split:Nee
3262     \seq_set_split:Nnon
3263     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3264 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3265 \tl_if_empty:NF \l_tmpa_tl
3266 { \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.

```

3267 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3268 { \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`).

```

3269 \tl_build_begin:N \l_@@_new_body_tl
3270 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3271 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3272 \@@_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).

```

3273 \seq_map_inline:Nn \l_@@_rows_seq
3274 {
3275   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3276   \@@_line_with_light_syntax:n { ##1 }
3277 }
3278 \tl_build_end:N \l_@@_new_body_tl
3279 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3280 {
3281   \int_set:Nn \l_@@_last_col_int
3282   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3283 }

```

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.

```

3284 \@@_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`).

```

3285 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3286 }
3287 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3288 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3289 {
3290   \seq_clear_new:N \l_@@_cells_seq
3291   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3292   \int_set:Nn \l_@@_nb_cols_int
3293   {
3294     \int_max:nn
3295     \l_@@_nb_cols_int
3296     { \seq_count:N \l_@@_cells_seq }
3297   }
3298   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3299   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3300   \seq_map_inline:Nn \l_@@_cells_seq
3301   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3302 }

```

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.

```

3303 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3304 {
3305   \str_if_eq:eeT \g_@@_name_env_str { #2 }
3306   { \@@_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.

```

3307   \end { #2 }
3308 }

```

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).

```

3309 \cs_new:Npn \@@_create_col_nodes:
3310 {
3311   \crrc
3312   \int_if_zero:nT \l_@@_first_col_int
3313   {
3314     \omit
3315     \hbox_overlap_left:n
3316     {
3317       \bool_if:NT \l_@@_code_before_bool
3318       { \pgfsys@markposition { \@@_env: - col - 0 } }
3319       \pgfpicture
3320       \pgfrememberpicturepositiononpagetrue
3321       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3322       \str_if_empty:NF \l_@@_name_str
3323       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3324       \endpgfpicture
3325       \skip_horizontal:N 2\col@sep
3326       \skip_horizontal:N \g_@@_width_first_col_dim
3327     }
3328     &
3329   }
3330   \omit

```

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

```

3331   \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).

```

3332   \int_if_zero:nTF \l_@@_first_col_int
3333   {
3334     \bool_if:NT \l_@@_code_before_bool
3335     {
3336       \hbox
3337       {
3338         \skip_horizontal:N -0.5\arrayrulewidth
3339         \pgfsys@markposition { \@@_env: - col - 1 }
3340         \skip_horizontal:N 0.5\arrayrulewidth
3341       }
3342     }
3343     \pgfpicture
3344     \pgfrememberpicturepositiononpagetrue
3345     \pgfcoordinate { \@@_env: - col - 1 }
3346     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3347     \str_if_empty:NF \l_@@_name_str
3348     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3349     \endpgfpicture
3350   }

```

```

3351 {
3352   \bool_if:NT \l_@@_code_before_bool
3353   {
3354     \hbox
3355     {
3356       \skip_horizontal:N 0.5\arrayrulewidth
3357       \pgfsys@markposition { \@@_env: - col - 1 }
3358       \skip_horizontal:N -0.5\arrayrulewidth
3359     }
3360   }
3361   \pgfpicture
3362   \pgfrememberpicturerepositiononpagetrue
3363   \pgfcoordinate { \@@_env: - col - 1 }
3364   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3365   \str_if_empty:NF \l_@@_name_str
3366   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3367   \endpgfpicture
3368 }

```

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.

```

3369 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3370 \bool_if:NF \l_@@_auto_columns_width_bool
3371 { \dim_compare:nNt \l_@@_columns_width_dim > \c_zero_dim }
3372 {
3373   \bool_lazy_and:nnTF
3374   \l_@@_auto_columns_width_bool
3375   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3376   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3377   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3378   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3379 }
3380 \skip_horizontal:N \g_tmpa_skip
3381 \hbox
3382 {
3383   \bool_if:NT \l_@@_code_before_bool
3384   {
3385     \hbox
3386     {
3387       \skip_horizontal:N -0.5\arrayrulewidth
3388       \pgfsys@markposition { \@@_env: - col - 2 }
3389       \skip_horizontal:N 0.5\arrayrulewidth
3390     }
3391   }
3392   \pgfpicture
3393   \pgfrememberpicturerepositiononpagetrue
3394   \pgfcoordinate { \@@_env: - col - 2 }
3395   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3396   \str_if_empty:NF \l_@@_name_str
3397   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3398   \endpgfpicture
3399 }

```

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.

```

3400 \int_gset_eq:NN \g_tmpa_int \c_one_int
3401 \bool_if:NTF \g_@@_last_col_found_bool
3402 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3403 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3404 {

```

```

3405      &
3406      \omit
3407      \int_gincr:N \g_tmpa_int

```

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

```

3408      \skip_horizontal:N \g_tmpa_skip
3409      \bool_if:NT \l_@@_code_before_bool
3410      {
3411        \hbox
3412        {
3413          \skip_horizontal:N -0.5\arrayrulewidth
3414          \pgfsys@markposition
3415          { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3416          \skip_horizontal:N 0.5\arrayrulewidth
3417        }
3418      }

```

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

```

3419      \pgfpicture
3420      \pgfrememberpicturepositiononpagetrue
3421      \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3422      { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3423      \str_if_empty:NF \l_@@_name_str
3424      {
3425        \pgfnodealias
3426        { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3427        { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3428      }
3429      \endpgfpicture
3430    }

```

```

3431      &
3432      \omit

```

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

```

3433      \int_if_zero:nT \g_@@_col_total_int
3434      { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3435      \skip_horizontal:N \g_tmpa_skip
3436      \int_gincr:N \g_tmpa_int
3437      \bool_lazy_any:nF
3438      {
3439        \g_@@_delims_bool
3440        \l_@@_tabular_bool
3441        { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3442        \l_@@_exterior_arraycolsep_bool
3443        \l_@@_bar_at_end_of_pream_bool
3444      }
3445      { \skip_horizontal:N -\col@sep }
3446      \bool_if:NT \l_@@_code_before_bool
3447      {
3448        \hbox
3449        {
3450          \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.

```

3451        \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3452        { \skip_horizontal:N -\arraycolsep }
3453        \pgfsys@markposition
3454        { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3455        \skip_horizontal:N 0.5\arrayrulewidth
3456        \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3457         { \skip_horizontal:N \arraycolsep }
3458     }
3459 }
3460 \pgfpicture
3461 \pgfrememberpicturepositiononpagetrue
3462 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3463 {
3464     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3465     {
3466         \pgfpoint
3467         { - 0.5 \arrayrulewidth - \arraycolsep }
3468         \c_zero_dim
3469     }
3470     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3471 }
3472 \str_if_empty:NF \l_@@_name_str
3473 {
3474     \pgfnodealias
3475     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3476     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3477 }
3478 \endpgfpicture

3479 \bool_if:NT \g_@@_last_col_found_bool
3480 {
3481     \hbox_overlap_right:n
3482     {
3483         \skip_horizontal:N \g_@@_width_last_col_dim
3484         \skip_horizontal:N \col@sep
3485         \bool_if:NT \l_@@_code_before_bool
3486         {
3487             \pgfsys@markposition
3488             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3489         }
3490         \pgfpicture
3491         \pgfrememberpicturepositiononpagetrue
3492         \pgfcoordinate
3493         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3494         \pgfpointorigin
3495         \str_if_empty:NF \l_@@_name_str
3496         {
3497             \pgfnodealias
3498             {
3499                 \l_@@_name_str - col
3500                 - \int_eval:n { \g_@@_col_total_int + 1 }
3501             }
3502             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3503         }
3504         \endpgfpicture
3505     }
3506 }
3507 % \cr
3508 }

```

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

```

3509 \tl_const:Nn \c_@@_preamble_first_col_tl
3510 {
3511     >
3512     {

```

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


```

3513 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3514 \bool_gset_true:N \g_@@_after_col_zero_bool
3515 \@@_begin_of_row:
3516 \hbox_set:Nw \l_@@_cell_box
3517 \@@_math_toggle:
3518 \@@_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”.

```

3519 \int_compare:nNnT \c@iRow > \c_zero_int
3520 {
3521   \bool_lazy_or:nnT
3522     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3523     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3524     {
3525       \l_@@_code_for_first_col_tl
3526       \xglobal \colorlet { nicematrix-first-col } { . }
3527     }
3528 }
3529 }

```

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`.

```

3530 1
3531 <
3532 {
3533   \@@_math_toggle:
3534   \hbox_set_end:
3535   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3536   \@@_adjust_size_box:
3537   \@@_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.

```

3538 \dim_gset:Nn \g_@@_width_first_col_dim
3539 { \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.

```

3540 \hbox_overlap_left:n
3541 {
3542   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3543     \@@_node_for_cell:
3544     { \box_use_drop:N \l_@@_cell_box }
3545     \skip_horizontal:N \l_@@_left_delim_dim
3546     \skip_horizontal:N \l_@@_left_margin_dim
3547     \skip_horizontal:N \l_@@_extra_left_margin_dim
3548   }
3549   \bool_gset_false:N \g_@@_empty_cell_bool
3550   \skip_horizontal:N -2\col@sep
3551 }
3552 }

```

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

```

3553 \tl_const:Nn \c_@@_preamble_last_col_tl
3554 {
3555   >
3556   {
3557     \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).

```

3558 \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.

```

3559 \bool_gset_true:N \g_@@_last_col_found_bool
3560 \int_gincr:N \c@jCol

```

```

3561 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
3562 \hbox_set:Nw \l_@@_cell_box
3563 \@@_math_toggle:
3564 \@@_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”.

```

3565 \int_compare:nNnT \c@iRow > \c_zero_int
3566 {
3567   \bool_lazy_or:nnT
3568   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3569   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3570   {
3571     \l_@@_code_for_last_col_tl
3572     \xglobal \colorlet { nicematrix-last-col } { . }
3573   }
3574 }
3575 }
3576 1
3577 <
3578 {
3579   \@@_math_toggle:
3580   \hbox_set_end:
3581   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3582   \@@_adjust_size_box:
3583   \@@_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.

```

3584 \dim_gset:Nn \g_@@_width_last_col_dim
3585 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3586 \skip_horizontal:N -2\col@sep

```

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

```

3587 \hbox_overlap_right:n
3588 {
3589   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3590   {
3591     \skip_horizontal:N \l_@@_right_delim_dim
3592     \skip_horizontal:N \l_@@_right_margin_dim
3593     \skip_horizontal:N \l_@@_extra_right_margin_dim
3594     \@@_node_for_cell:
3595   }
3596 }
3597 \bool_gset_false:N \g_@@_empty_cell_bool
3598 }
3599 }

```

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

```

3600 \NewDocumentEnvironment { NiceArray } { }
3601 {
3602   \bool_gset_false:N \g_@@_delims_bool
3603   \str_if_empty:NT \g_@@_name_env_str
3604   { \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).

```

3605 \NiceArrayWithDelims . .
3606 }
3607 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

3608 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3609 {
3610   \NewDocumentEnvironment { #1 NiceArray } { }
3611   {
3612     \bool_gset_true:N \g_@@_delims_bool
3613     \str_if_empty:NT \g_@@_name_env_str
3614     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3615     \@@_test_if_math_mode:
3616     \NiceArrayWithDelims #2 #3
3617   }
3618   { \endNiceArrayWithDelims }
3619 }
3620 \@@_def_env:nnn p ( )
3621 \@@_def_env:nnn b [ ]
3622 \@@_def_env:nnn B \{ \}
3623 \@@_def_env:nnn v | |
3624 \@@_def_env:nnn V \l \r

```

13 The environment `{NiceMatrix}` and its variants

```

3625 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3626 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3627 {
3628   \bool_set_false:N \l_@@_preamble_bool
3629   \tl_clear:N \l_tmpa_tl
3630   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3631   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3632   \tl_put_right:Nn \l_tmpa_tl
3633   {
3634     *
3635     {
3636       \int_case:nnF \l_@@_last_col_int
3637       {
3638         { -2 } { \c@MaxMatrixCols }
3639         { -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).

```

3640       }
3641       { \int_eval:n { \l_@@_last_col_int - 1 } }
3642     }
3643     { #2 }
3644   }
3645   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3646   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3647 }
3648 \clist_map_inline:nn { p , b , B , v , V }
3649 {
3650   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3651   {
3652     \bool_gset_true:N \g_@@_delims_bool
3653     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3654     \int_if_zero:nT \l_@@_last_col_int
3655     {
3656       \bool_set_true:N \l_@@_last_col_without_value_bool
3657       \int_set:Nn \l_@@_last_col_int { -1 }
3658     }
3659     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3660     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3661   }

```

```

3662     { \use:c { end #1 NiceArray } }
3663 }

```

We define also an environment {NiceMatrix}

```

3664 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3665 {
3666     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3667     \int_if_zero:nT \l_@@_last_col_int
3668     {
3669         \bool_set_true:N \l_@@_last_col_without_value_bool
3670         \int_set:Nn \l_@@_last_col_int { -1 }
3671     }
3672     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3673     \bool_lazy_or:nnT
3674     { \clist_if_empty_p:N \l_@@_vlines_clist }
3675     { \l_@@_except_borders_bool }
3676     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3677     \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3678 }
3679 { \endNiceArray }

```

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

```

3680 \cs_new_protected:Npn \@@_NotEmpty:
3681 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3682 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3683 {

```

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.

```

3684     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3685     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3686     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3687     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3688     \tl_if_empty:NF \l_@@_short_caption_tl
3689     {
3690         \tl_if_empty:NT \l_@@_caption_tl
3691         {
3692             \@@_error_or_warning:n { short-caption-without~caption }
3693             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3694         }
3695     }
3696     \tl_if_empty:NF \l_@@_label_tl
3697     {
3698         \tl_if_empty:NT \l_@@_caption_tl
3699         { \@@_error_or_warning:n { label~without~caption } }
3700     }
3701     \NewDocumentEnvironment { TabularNote } { b }
3702     {
3703         \bool_if:NTF \l_@@_in_code_after_bool
3704         { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3705         {
3706             \tl_if_empty:NF \g_@@_tabularnote_tl
3707             { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3708             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3709         }
3710     }
3711     { }
3712     \@@_settings_for_tabular:

```

```

3713 \NiceArray { #2 }
3714 }
3715 { \endNiceArray }
3716 \cs_new_protected:Npn \@@_settings_for_tabular:
3717 {
3718 \bool_set_true:N \l_@@_tabular_bool
3719 \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3720 \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3721 \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3722 }

3723 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3724 {
3725 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3726 \dim_zero_new:N \l_@@_width_dim
3727 \dim_set:Nn \l_@@_width_dim { #1 }
3728 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3729 \@@_settings_for_tabular:
3730 \NiceArray { #3 }
3731 }
3732 {
3733 \endNiceArray
3734 \int_if_zero:nT \g_@@_total_X_weight_int
3735 { \@@_error:n { NiceTabularX~without~X } }
3736 }

3737 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3738 {
3739 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3740 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3741 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3742 \@@_settings_for_tabular:
3743 \NiceArray { #3 }
3744 }
3745 { \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`).

```

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

```

```

3764         draw = \l_@@_rules_color_tl
3765     }
3766     { 1-1 }
3767     { \int_use:N \c@iRow - \int_use:N \c@jCol }
3768 }
3769 }
3770 }

```

```

3771 \cs_new_protected:Npn \@@_after_array:
3772 {

```

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`.

```

3773     \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3774     \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.

```

3775     \bool_if:NT \g_@@_last_col_found_bool
3776     { \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`.

```

3777     \bool_if:NT \l_@@_last_col_without_value_bool
3778     { \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.

```

3779     \bool_if:NT \l_@@_last_row_without_value_bool
3780     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3781     \tl_gput_right:Ne \g_@@_aux_tl
3782     {
3783         \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3784         {
3785             \int_use:N \l_@@_first_row_int ,
3786             \int_use:N \c@iRow ,
3787             \int_use:N \g_@@_row_total_int ,
3788             \int_use:N \l_@@_first_col_int ,
3789             \int_use:N \c@jCol ,
3790             \int_use:N \g_@@_col_total_int
3791         }
3792     }

```

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`).

```

3793     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3794     {
3795         \tl_gput_right:Ne \g_@@_aux_tl
3796         {
3797             \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3798             { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3799         }
3800     }
3801     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3802     {
3803         \tl_gput_right:Ne \g_@@_aux_tl

```

```

3804     {
3805         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3806         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3807         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3808         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3809     }
3810 }

```

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

```

3811 \@@_create_diag_nodes:

```

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

```

3812 \pgfpicture
3813 \int_step_inline:nn \c@iRow
3814 {
3815     \pgfnodealias
3816     { \@@_env: - ##1 - last }
3817     { \@@_env: - ##1 - \int_use:N \c@jCol }
3818 }
3819 \int_step_inline:nn \c@jCol
3820 {
3821     \pgfnodealias
3822     { \@@_env: - last - ##1 }
3823     { \@@_env: - \int_use:N \c@iRow - ##1 }
3824 }
3825 \str_if_empty:NF \l_@@_name_str
3826 {
3827     \int_step_inline:nn \c@iRow
3828     {
3829         \pgfnodealias
3830         { \l_@@_name_str - ##1 - last }
3831         { \@@_env: - ##1 - \int_use:N \c@jCol }
3832     }
3833     \int_step_inline:nn \c@jCol
3834     {
3835         \pgfnodealias
3836         { \l_@@_name_str - last - ##1 }
3837         { \@@_env: - \int_use:N \c@iRow - ##1 }
3838     }
3839 }
3840 \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.

```

3841 \bool_if:NT \l_@@_parallelize_diags_bool
3842 {
3843     \int_gzero_new:N \g_@@_ddots_int
3844     \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.

```

3845     \dim_gzero_new:N \g_@@_delta_x_one_dim
3846     \dim_gzero_new:N \g_@@_delta_y_one_dim
3847     \dim_gzero_new:N \g_@@_delta_x_two_dim
3848     \dim_gzero_new:N \g_@@_delta_y_two_dim
3849 }

```

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

```

3850 \int_zero_new:N \l_@@_initial_i_int
3851 \int_zero_new:N \l_@@_initial_j_int
3852 \int_zero_new:N \l_@@_final_i_int
3853 \int_zero_new:N \l_@@_final_j_int
3854 \bool_set_false:N \l_@@_initial_open_bool
3855 \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.

```

3856 \bool_if:NT \l_@@_small_bool
3857 {
3858   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3859   \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.

```

3860   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3861     { 0.6 \l_@@_xdots_shorten_start_dim }
3862   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3863     { 0.6 \l_@@_xdots_shorten_end_dim }
3864 }

```

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

```

3865 \@@_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.

```

3866 \clist_if_empty:NF \l_@@_corners_clist \@@_compute_corners:

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3867 \@@_adjust_pos_of_blocks_seq:
3868 \@@_deal_with_rounded_corners:
3869 \clist_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3870 \clist_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3871 \IfPackageLoadedT { tikz }
3872 {
3873   \tikzset
3874   {
3875     every-picture / .style =
3876     {
3877       overlay ,
3878       remember~picture ,
3879       name~prefix = \@@_env: -
3880     }
3881   }
3882 }
3883 \bool_if:NT \c_@@_recent_array_bool
3884 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3885 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3886 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3887 \cs_set_eq:NN \OverBrace \@@_OverBrace
3888 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3889 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3890 \cs_set_eq:NN \line \@@_line
3891 \g_@@_pre_code_after_tl
3892 \tl_gclear:N \g_@@_pre_code_after_tl

```


When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it's possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That's why we set `\Code-after` to be *no-op* now.

```
3893 \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).

```
3894 \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`).

```
3895 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3896 { \@@_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:`.

```
3897 \bool_set_true:N \l_@@_in_code_after_bool
3898 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3899 \scan_stop:
3900 \tl_gclear:N \g_nicematrix_code_after_tl
3901 \group_end:
```

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

```
3902 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3903 \tl_if_empty:NF \g_@@_pre_code_before_tl
3904 {
3905   \tl_gput_right:Ne \g_@@_aux_tl
3906   {
3907     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3908     { \exp_not:o \g_@@_pre_code_before_tl }
3909   }
3910   \tl_gclear:N \g_@@_pre_code_before_tl
3911 }
3912 \tl_if_empty:NF \g_nicematrix_code_before_tl
3913 {
3914   \tl_gput_right:Ne \g_@@_aux_tl
3915   {
3916     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3917     { \exp_not:o \g_nicematrix_code_before_tl }
3918   }
3919   \tl_gclear:N \g_nicematrix_code_before_tl
3920 }

3921 \str_gclear:N \g_@@_name_env_str
3922 \@@_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`.

```
3923 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3924 }
```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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`.

```

3925 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3926 { \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).

```

3927 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3928 {
3929   \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3930   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3931 }

```

The following command must *not* be protected.

```

3932 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3933 {
3934   { #1 }
3935   { #2 }
3936   {
3937     \int_compare:nNnTF { #3 } > { 99 }
3938     { \int_use:N \c@iRow }
3939     { #3 }
3940   }
3941   {
3942     \int_compare:nNnTF { #4 } > { 99 }
3943     { \int_use:N \c@jCol }
3944     { #4 }
3945   }
3946   { #5 }
3947 }

```

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).

```

3948 \hook_gput_code:nnn { begindocument } { . }
3949 {
3950   \cs_new_protected:Npe \@@_draw_dotted_lines:
3951   {
3952     \c_@@_pgfortikzpicture_tl
3953     \@@_draw_dotted_lines_i:
3954     \c_@@_endpgfortikzpicture_tl
3955   }
3956 }

```

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

```

3957 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3958 {
3959   \pgfrememberpicturepositiononpagetrue
3960   \pgf@relevantforpicturesizefalse
3961   \g_@@_HVdotsfor_lines_tl
3962   \g_@@_Vdots_lines_tl
3963   \g_@@_Ddots_lines_tl
3964   \g_@@_Idots_lines_tl
3965   \g_@@_Cdots_lines_tl
3966   \g_@@_Ldots_lines_tl
3967 }

```

```

3968 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3969 {
3970   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3971   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3972 }

```

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

```

3973 \pgfdeclareshape { @@_diag_node }
3974 {
3975   \savedanchor { \five }
3976   {
3977     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3978     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3979   }
3980   \anchor { 5 } { \five }
3981   \anchor { center } { \pgfpointorigin }
3982   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
3983   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
3984   \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
3985   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
3986   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
3987   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
3988   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
3989   \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
3990   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
3991   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
3992 }

```

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).

```

3993 \cs_new_protected:Npn \@@_create_diag_nodes:
3994 {
3995   \pgfpicture
3996   \pgfrememberpicturepositiononpagetrue
3997   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3998   {
3999     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4000     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4001     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4002     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4003     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4004     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4005     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4006     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4007     \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.

```

4008     \dim_set:Nn \l_tmpa_int { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4009     \dim_set:Nn \l_tmpb_int { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4010     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4011     \str_if_empty:NF \l_@@_name_str
4012     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4013   }

```

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

```

4014     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4015     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4016     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4017     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4018     \pgfcoordinate

```

```

4019     { \l_@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4020 \pgfnodealias
4021   { \l_@@_env: - last }
4022   { \l_@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4023 \str_if_empty:NF \l_@@_name_str
4024   {
4025     \pgfnodealias
4026       { \l_@@_name_str - \int_use:N \l_tmpa_int }
4027       { \l_@@_env: - \int_use:N \l_tmpa_int }
4028     \pgfnodealias
4029       { \l_@@_name_str - last }
4030       { \l_@@_env: - last }
4031   }
4032 \endpgfpicture
4033 }

```

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 `\l_@@_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.

```

4034 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4035   {

```

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

```

4036   \cs_set_nopar:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4037   \int_set:Nn \l_@@_initial_i_int { #1 }
4038   \int_set:Nn \l_@@_initial_j_int { #2 }
4039   \int_set:Nn \l_@@_final_i_int { #1 }
4040   \int_set:Nn \l_@@_final_j_int { #2 }

```

We will do two loops: one when determinating the initial cell and the other when determinating 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.

```

4041     \bool_set_false:N \l_@@_stop_loop_bool
4042     \bool_do_until:Nn \l_@@_stop_loop_bool
4043     {
4044         \int_add:Nn \l_@@_final_i_int { #3 }
4045         \int_add:Nn \l_@@_final_j_int { #4 }
4046         \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.

```

4047     \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4048     \if_int_compare:w #3 = \c_one_int
4049         \bool_set_true:N \l_@@_final_open_bool
4050     \else:
4051         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4052             \bool_set_true:N \l_@@_final_open_bool
4053         \fi:
4054     \fi:
4055 \else:
4056     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4057         \if_int_compare:w #4 = -1
4058             \bool_set_true:N \l_@@_final_open_bool
4059         \fi:
4060     \else:
4061         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4062             \if_int_compare:w #4 = \c_one_int
4063                 \bool_set_true:N \l_@@_final_open_bool
4064             \fi:
4065         \fi:
4066     \fi:
4067 \fi:
4068     \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.

```

4069     {

```

We do a step backwards.

```

4070         \int_sub:Nn \l_@@_final_i_int { #3 }
4071         \int_sub:Nn \l_@@_final_j_int { #4 }
4072         \bool_set_true:N \l_@@_stop_loop_bool
4073     }

```

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`.

```

4074     {
4075         \cs_if_exist:cTF
4076         {
4077             @@ _ dotted _
4078             \int_use:N \l_@@_final_i_int -
4079             \int_use:N \l_@@_final_j_int
4080         }
4081         {
4082             \int_sub:Nn \l_@@_final_i_int { #3 }
4083             \int_sub:Nn \l_@@_final_j_int { #4 }
4084             \bool_set_true:N \l_@@_final_open_bool
4085             \bool_set_true:N \l_@@_stop_loop_bool
4086         }
4087     }
4088     \cs_if_exist:cTF
4089     {
4090         pgf @ sh @ ns @ \@@_env:
4091         - \int_use:N \l_@@_final_i_int

```

```

4092         - \int_use:N \l_@@_final_j_int
4093     }
4094     { \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.

```

4095     {
4096         \cs_set_nopar:cpn
4097         {
4098             @@ _ dotted _
4099             \int_use:N \l_@@_final_i_int -
4100             \int_use:N \l_@@_final_j_int
4101         }
4102         { }
4103     }
4104 }
4105 }
4106 }

```

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

```

4107     \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4108     \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4109     \bool_do_until:Nn \l_@@_stop_loop_bool
4110     {
4111         \int_sub:Nn \l_@@_initial_i_int { #3 }
4112         \int_sub:Nn \l_@@_initial_j_int { #4 }
4113         \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.

```

4114         \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4115         \if_int_compare:w #3 = \c_one_int
4116         \bool_set_true:N \l_@@_initial_open_bool
4117         \else:

```

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

```

4118         \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4119         \bool_set_true:N \l_@@_initial_open_bool
4120         \fi:
4121         \fi:
4122         \else:
4123         \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4124         \if_int_compare:w #4 = \c_one_int
4125         \bool_set_true:N \l_@@_initial_open_bool
4126         \fi:
4127         \else:
4128         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4129         \if_int_compare:w #4 = -1
4130         \bool_set_true:N \l_@@_initial_open_bool
4131         \fi:
4132         \fi:
4133         \fi:
4134         \fi:

```

```

4135 \bool_if:NTF \l_@@_initial_open_bool
4136 {
4137   \int_add:Nn \l_@@_initial_i_int { #3 }
4138   \int_add:Nn \l_@@_initial_j_int { #4 }
4139   \bool_set_true:N \l_@@_stop_loop_bool
4140 }
4141 {
4142   \cs_if_exist:cTF
4143   {
4144     @@ _ dotted _
4145     \int_use:N \l_@@_initial_i_int -
4146     \int_use:N \l_@@_initial_j_int
4147   }
4148   {
4149     \int_add:Nn \l_@@_initial_i_int { #3 }
4150     \int_add:Nn \l_@@_initial_j_int { #4 }
4151     \bool_set_true:N \l_@@_initial_open_bool
4152     \bool_set_true:N \l_@@_stop_loop_bool
4153   }
4154   {
4155     \cs_if_exist:cTF
4156     {
4157       pgf @ sh @ ns @ \@@_env:
4158       - \int_use:N \l_@@_initial_i_int
4159       - \int_use:N \l_@@_initial_j_int
4160     }
4161     { \bool_set_true:N \l_@@_stop_loop_bool }
4162     {
4163       \cs_set_nopar:cpn
4164       {
4165         @@ _ dotted _
4166         \int_use:N \l_@@_initial_i_int -
4167         \int_use:N \l_@@_initial_j_int
4168       }
4169       { }
4170     }
4171   }
4172 }
4173 }

```

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.

```

4174 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4175 {
4176   { \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`.

```

4177 { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4178 { \int_use:N \l_@@_final_i_int }
4179 { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4180 { } % for the name of the block
4181 }
4182 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analyse of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4183 \cs_new_protected:Npn \@@_open_shorten:
4184 {

```

```

4185 \bool_if:NT \l_@@_initial_open_bool
4186 { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4187 \bool_if:NT \l_@@_final_open_bool
4188 { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4189 }

```

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).

```

4190 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4191 {
4192   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4193   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4194   \int_set_eq:NN \l_@@_row_max_int \c_iRow
4195   \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`.

```

4196 \seq_if_empty:NF \g_@@_submatrix_seq
4197 {
4198   \seq_map_inline:Nn \g_@@_submatrix_seq
4199   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4200 }
4201 }

```

#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 programmation of that command with the the standard syntax of L3.

```

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

```

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

```

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

```



```

4216         \fi:
4217     \fi:
4218     \fi:
4219     \fi:
4220 }

4221 \cs_new_protected:Npn \@@_set_initial_coords:
4222 {
4223     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4224     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4225 }
4226 \cs_new_protected:Npn \@@_set_final_coords:
4227 {
4228     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4229     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4230 }
4231 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4232 {
4233     \pgfpointanchor
4234     {
4235         \@@_env:
4236         - \int_use:N \l_@@_initial_i_int
4237         - \int_use:N \l_@@_initial_j_int
4238     }
4239     { #1 }
4240     \@@_set_initial_coords:
4241 }
4242 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4243 {
4244     \pgfpointanchor
4245     {
4246         \@@_env:
4247         - \int_use:N \l_@@_final_i_int
4248         - \int_use:N \l_@@_final_j_int
4249     }
4250     { #1 }
4251     \@@_set_final_coords:
4252 }

4253 \cs_new_protected:Npn \@@_open_x_initial_dim:
4254 {
4255     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4256     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4257     {
4258         \cs_if_exist:cT
4259         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4260         {
4261             \pgfpointanchor
4262             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4263             { west }
4264             \dim_set:Nn \l_@@_x_initial_dim
4265             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4266         }
4267     }

```

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

```

4268     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4269     {
4270         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4271         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4272         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4273     }
4274 }

```

```

4275 \cs_new_protected:Npn \@@_open_x_final_dim:
4276 {
4277   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4278   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4279   {
4280     \cs_if_exist:cT
4281     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4282     {
4283       \pgfpointanchor
4284       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4285       { east }
4286       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4287       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4288     }
4289   }

```

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

```

4290   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4291   {
4292     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4293     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4294     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4295   }
4296 }

```

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.

```

4297 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4298 {
4299   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4300   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4301   {
4302     \@@_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.

```

4303   \group_begin:
4304   \@@_open_shorten:
4305   \int_if_zero:nTF { #1 }
4306   { \color { nicematrix-first-row } }
4307   {

```

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.

```

4308       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4309       { \color { nicematrix-last-row } }
4310     }
4311     \keys_set:nn { nicematrix / xdots } { #3 }
4312     \@@_color:o \l_@@_xdots_color_tl
4313     \@@_actually_draw_Ldots:
4314   \group_end:
4315 }
4316 }

```

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.

```

4317 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4318 {
4319   \bool_if:NTF \l_@@_initial_open_bool
4320   {
4321     \@@_open_x_initial_dim:
4322     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4323     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4324   }
4325   { \@@_set_initial_coords_from_anchor:n { base-east } }
4326   \bool_if:NTF \l_@@_final_open_bool
4327   {
4328     \@@_open_x_final_dim:
4329     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4330     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4331   }
4332   { \@@_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.

```

4333   \bool_lazy_all:nTF
4334   {
4335     \l_@@_initial_open_bool
4336     \l_@@_final_open_bool
4337     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4338   }
4339   {
4340     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4341     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4342   }

```

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 (?).

```

4343   {
4344     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4345     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4346   }
4347   \@@_draw_line:
4348 }

```

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.

```

4349 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4350 {
4351   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4352   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4353   {
4354     \@@_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.

```

4355   \group_begin:
4356   \@@_open_shorten:
4357   \int_if_zero:nTF { #1 }
4358   { \color { nicematrix-first-row } }
4359   {

```

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.

```

4360         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4361         { \color { nicematrix-last-row } }
4362     }
4363     \keys_set:nn { nicematrix / xdots } { #3 }
4364     \@@_color:o \l_@@_xdots_color_tl
4365     \@@_actually_draw_Cdots:
4366 \group_end:
4367 }
4368 }

```

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.`

```

4369 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4370 {
4371     \bool_if:NTF \l_@@_initial_open_bool
4372     { \@@_open_x_initial_dim: }
4373     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4374     \bool_if:NTF \l_@@_final_open_bool
4375     { \@@_open_x_final_dim: }
4376     { \@@_set_final_coords_from_anchor:n { mid-west } }
4377     \bool_lazy_and:nnTF
4378     \l_@@_initial_open_bool
4379     \l_@@_final_open_bool
4380     {
4381         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4382         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4383         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4384         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4385         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4386     }
4387     {
4388         \bool_if:NT \l_@@_initial_open_bool
4389         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4390         \bool_if:NT \l_@@_final_open_bool
4391         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4392     }
4393     \@@_draw_line:
4394 }
4395 \cs_new_protected:Npn \@@_open_y_initial_dim:
4396 {
4397     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4398     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4399     {
4400         \cs_if_exist:cT
4401         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4402         {
4403             \pgfpointanchor
4404             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4405             { north }
4406             \dim_compare:nNnT \pgf@y > \l_@@_y_initial_dim

```

```

4407         { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
4408     }
4409 }
4410 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4411 {
4412     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4413     \dim_set:Nn \l_@@_y_initial_dim
4414     {
4415         \fp_to_dim:n
4416         {
4417             \pgf@y
4418             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4419         }
4420     }
4421 }
4422 }
4423 \cs_new_protected:Npn \@@_open_y_final_dim:
4424 {
4425     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4426     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4427     {
4428         \cs_if_exist:cT
4429         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4430         {
4431             \pgfpointanchor
4432             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4433             { south }
4434             \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
4435             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
4436         }
4437     }
4438     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4439     {
4440         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4441         \dim_set:Nn \l_@@_y_final_dim
4442         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4443     }
4444 }

```

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.

```

4445 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4446 {
4447     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4448     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4449     {
4450         \@@_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.

```

4451     \group_begin:
4452     \@@_open_shorten:
4453     \int_if_zero:nTF { #2 }
4454     { \color { nicematrix-first-col } }
4455     {
4456         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4457         { \color { nicematrix-last-col } }
4458     }
4459     \keys_set:nn { nicematrix / xdots } { #3 }
4460     \@@_color:o \l_@@_xdots_color_tl
4461     \@@_actually_draw_Vdots:
4462     \group_end:
4463 }
4464 }

```

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`.

```
4465 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4466 {
```

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

```
4467 \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.

```
4468 {
4469   \@@_open_y_initial_dim:
4470   \@@_open_y_final_dim:
4471   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4472 {
4473   \@@_qpoint:n { col - 1 }
4474   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4475   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4476   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4477   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4478 }
4479 {
4480   \bool_lazy_and:nnTF
4481   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4482   { \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”.

```
4483 {
4484   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4485   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4486   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4487   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4488   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4489 }
```

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

```
4490 {
4491   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4492   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4493   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4494   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4495 }
4496 }
4497 }
```

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.

```
4498 {
4499   \bool_set_false:N \l_tmpa_bool
4500   \bool_if:NF \l_@@_initial_open_bool
4501   {
4502     \bool_if:NF \l_@@_final_open_bool
4503     {
```

```

4504         \@@_set_initial_coords_from_anchor:n { south-west }
4505         \@@_set_final_coords_from_anchor:n { north-west }
4506         \bool_set:Nn \l_tmpa_bool
4507         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4508     }
4509 }

```

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

```

4510     \bool_if:NTF \l_@@_initial_open_bool
4511     {
4512         \@@_open_y_initial_dim:
4513         \@@_set_final_coords_from_anchor:n { north }
4514         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4515     }
4516     {
4517         \@@_set_initial_coords_from_anchor:n { south }
4518         \bool_if:NTF \l_@@_final_open_bool
4519         \@@_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.

```

4520     {
4521         \@@_set_final_coords_from_anchor:n { north }
4522         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4523         {
4524             \dim_set:Nn \l_@@_x_initial_dim
4525             {
4526                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4527                 \l_@@_x_initial_dim \l_@@_x_final_dim
4528             }
4529         }
4530     }
4531 }
4532 }
4533 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4534 \@@_draw_line:
4535 }

```

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.

```

4536 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4537 {
4538     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4539     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4540     {
4541         \@@_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.

```

4542     \group_begin:
4543     \@@_open_shorten:
4544     \keys_set:nn { nicematrix / xdots } { #3 }
4545     \@@_color:o \l_@@_xdots_color_tl
4546     \@@_actually_draw_Ddots:
4547     \group_end:
4548 }
4549 }

```

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.

```

4550 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4551 {
4552   \bool_if:NTF \l_@@_initial_open_bool
4553   {
4554     \@@_open_y_initial_dim:
4555     \@@_open_x_initial_dim:
4556   }
4557   { \@@_set_initial_coords_from_anchor:n { south~east } }
4558   \bool_if:NTF \l_@@_final_open_bool
4559   {
4560     \@@_open_x_final_dim:
4561     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4562   }
4563   { \@@_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.

```

4564   \bool_if:NT \l_@@_parallelize_diags_bool
4565   {
4566     \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).

```

4567     \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.

```

4568     {
4569       \dim_gset:Nn \g_@@_delta_x_one_dim
4570       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4571       \dim_gset:Nn \g_@@_delta_y_one_dim
4572       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4573     }

```

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.

```

4574     {
4575       \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4576       {
4577         \dim_set:Nn \l_@@_y_final_dim
4578         {
4579           \l_@@_y_initial_dim +
4580           ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4581           \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4582         }
4583       }
4584     }
4585   }
4586   \@@_draw_line:
4587 }

```


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.

```

4588 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4589 {
4590   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4591   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4592   {
4593     \@@_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.

```

4594     \group_begin:
4595     \@@_open_shorten:
4596     \keys_set:nn { nicematrix / xdots } { #3 }
4597     \@@_color:o \l_@@_xdots_color_tl
4598     \@@_actually_draw_Iddots:
4599   \group_end:
4600 }
4601 }

```

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`.

```

4602 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4603 {
4604   \bool_if:NTF \l_@@_initial_open_bool
4605   {
4606     \@@_open_y_initial_dim:
4607     \@@_open_x_initial_dim:
4608   }
4609   { \@@_set_initial_coords_from_anchor:n { south-west } }
4610   \bool_if:NTF \l_@@_final_open_bool
4611   {
4612     \@@_open_y_final_dim:
4613     \@@_open_x_final_dim:
4614   }
4615   { \@@_set_final_coords_from_anchor:n { north-east } }
4616   \bool_if:NT \l_@@_parallelize_diags_bool
4617   {
4618     \int_gincr:N \g_@@_iddots_int
4619     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4620     {
4621       \dim_gset:Nn \g_@@_delta_x_two_dim
4622       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4623       \dim_gset:Nn \g_@@_delta_y_two_dim
4624       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4625     }
4626     {
4627       \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4628       {
4629         \dim_set:Nn \l_@@_y_final_dim
4630         {
4631           \l_@@_y_initial_dim +

```

```

4632         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4633         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4634     }
4635 }
4636 }
4637 }
4638 \@@_draw_line:
4639 }

```

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`

```

4640 \cs_new_protected:Npn \@@_draw_line:
4641 {
4642     \pgfrememberpicturepositiononpagetrue
4643     \pgf@relevantforpicturesizefalse
4644     \bool_lazy_or:nnTF
4645         { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4646         \l_@@_dotted_bool
4647         \@@_draw_standard_dotted_line:
4648         \@@_draw_unstandard_dotted_line:
4649 }

```

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.

```

4650 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4651 {
4652     \begin { scope }
4653     \@@_draw_unstandard_dotted_line:o
4654     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4655 }

```

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.

```

4656 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4657 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4658 {
4659     \@@_draw_unstandard_dotted_line:nooo
4660     { #1 }
4661     \l_@@_xdots_up_tl
4662     \l_@@_xdots_down_tl
4663     \l_@@_xdots_middle_tl
4664 }

```

The following Tikz styles are for the three labels (set by the symbols $_$, $^$ and $=$) of a continuous line with a non-standard style.

```

4665 \hook_gput_code:nnn { begindocument } { . }
4666 {
4667   \IfPackageLoadedT { tikz }
4668   {
4669     \tikzset
4670     {
4671       @@_node_above / .style = { sloped , above } ,
4672       @@_node_below / .style = { sloped , below } ,
4673       @@_node_middle / .style =
4674       {
4675         sloped ,
4676         inner-sep = \c_@@_innersep_middle_dim
4677       }
4678     }
4679   }
4680 }

4681 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4682 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4683 {

```

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.

```

4684 \dim_zero_new:N \l_@@_l_dim
4685 \dim_set:Nn \l_@@_l_dim
4686 {
4687   \fp_to_dim:n
4688   {
4689     sqrt
4690     (
4691       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4692       +
4693       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4694     )
4695   }
4696 }

```

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.

```

4697 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4698 {
4699   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4700   \@@_draw_unstandard_dotted_line_i:
4701 }

```

If the key `xdots/horizontal-labels` has been used.

```

4702 \bool_if:NT \l_@@_xdots_h_labels_bool
4703 {
4704   \tikzset
4705   {
4706     @@_node_above / .style = { auto = left } ,
4707     @@_node_below / .style = { auto = right } ,
4708     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4709   }
4710 }

```

```

4711 \tl_if_empty:nF { #4 }
4712 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4713 \draw
4714 [ #1 ]
4715 ( \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).

```

4716 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4717 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4718 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4719 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4720 \end { scope }
4721 }
4722 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4723 {
4724 \dim_set:Nn \l_tmpa_dim
4725 {
4726 \l_@@_x_initial_dim
4727 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4728 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4729 }
4730 \dim_set:Nn \l_tmpb_dim
4731 {
4732 \l_@@_y_initial_dim
4733 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4734 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4735 }
4736 \dim_set:Nn \l_@@_tmpc_dim
4737 {
4738 \l_@@_x_final_dim
4739 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4740 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4741 }
4742 \dim_set:Nn \l_@@_tmpd_dim
4743 {
4744 \l_@@_y_final_dim
4745 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4746 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4747 }
4748 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4749 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4750 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4751 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4752 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4753 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4754 {
4755 \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.

```

4756 \dim_zero_new:N \l_@@_l_dim
4757 \dim_set:Nn \l_@@_l_dim
4758 {
4759 \fp_to_dim:n
4760 {
4761 sqrt
4762 (
4763 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4764 +

```

```

4765         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4766     )
4767 }
4768 }

```

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.

```

4769 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4770 {
4771     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4772     \@@_draw_standard_dotted_line_i:
4773 }
4774 \group_end:
4775 \bool_lazy_all:nF
4776 {
4777     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4778     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4779     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4780 }
4781 \l_@@_labels_standard_dotted_line:
4782 }
4783 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4784 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4785 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4786 \int_set:Nn \l_tmpa_int
4787 {
4788     \dim_ratio:nn
4789     {
4790         \l_@@_l_dim
4791         - \l_@@_xdots_shorten_start_dim
4792         - \l_@@_xdots_shorten_end_dim
4793     }
4794     \l_@@_xdots_inter_dim
4795 }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4796 \dim_set:Nn \l_tmpa_dim
4797 {
4798     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4799     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4800 }
4801 \dim_set:Nn \l_tmpb_dim
4802 {
4803     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4804     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4805 }

```

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.

```

4806 \dim_gadd:Nn \l_@@_x_initial_dim
4807 {
4808     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4809     \dim_ratio:nn
4810     {
4811         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4812         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4813     }

```

```

4814         { 2 \l_@@_l_dim }
4815     }
4816     \dim_gadd:Nn \l_@@_y_initial_dim
4817     {
4818         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4819         \dim_ratio:nn
4820         {
4821             \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4822             + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4823         }
4824         { 2 \l_@@_l_dim }
4825     }
4826     \pgf@relevantforpicturesizefalse
4827     \int_step_inline:nnn \c_zero_int \l_tmpa_int
4828     {
4829         \pgfpathcircle
4830         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4831         { \l_@@_xdots_radius_dim }
4832         \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4833         \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4834     }
4835     \pgfusepathqfill
4836 }

4837 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4838 {
4839     \pgfscope
4840     \pgftransformshift
4841     {
4842         \pgfpointlineattime { 0.5 }
4843         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4844         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4845     }
4846     \fp_set:Nn \l_tmpa_fp
4847     {
4848         atand
4849         (
4850             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4851             \l_@@_x_final_dim - \l_@@_x_initial_dim
4852         )
4853     }
4854     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4855     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4856     \tl_if_empty:NF \l_@@_xdots_middle_tl
4857     {
4858         \begin { pgfscope }
4859         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4860         \pgfnode
4861         { rectangle }
4862         { center }
4863         {
4864             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4865             {
4866                 \c_math_toggle_token
4867                 \scriptstyle \l_@@_xdots_middle_tl
4868                 \c_math_toggle_token
4869             }
4870         }
4871         { }
4872         {
4873             \pgfsetfillcolor { white }
4874             \pgfusepath { fill }
4875         }

```

```

4876     \end { pgfscope }
4877   }
4878   \tl_if_empty:NF \l_@@_xdots_up_tl
4879   {
4880     \pgfnode
4881     { rectangle }
4882     { south }
4883     {
4884       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4885       {
4886         \c_math_toggle_token
4887         \scriptstyle \l_@@_xdots_up_tl
4888         \c_math_toggle_token
4889       }
4890     }
4891     { }
4892     { \pgfusepath { } }
4893   }
4894   \tl_if_empty:NF \l_@@_xdots_down_tl
4895   {
4896     \pgfnode
4897     { rectangle }
4898     { north }
4899     {
4900       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4901       {
4902         \c_math_toggle_token
4903         \scriptstyle \l_@@_xdots_down_tl
4904         \c_math_toggle_token
4905       }
4906     }
4907     { }
4908     { \pgfusepath { } }
4909   }
4910   \endpgfscope
4911 }

```

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.

```

4912 \hook_gput_code:nnn { begindocument } { . }
4913 {
4914   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4915   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4916   \cs_new_protected:Npn \@@_Ldots
4917   { \@@_collect_options:n { \@@_Ldots_i } }
4918   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4919   {
4920     \int_if_zero:nTF \c_jCol
4921     { \@@_error:nn { in~first~col } \Ldots }
4922     {

```

```

4923     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4924     { \@@_error:nn { in~last~col } \Ldots }
4925     {
4926         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4927         { #1 , down = #2 , up = #3 , middle = #4 }
4928     }
4929 }
4930 \bool_if:NF \l_@@_nullify_dots_bool
4931 { \phantom { \ensuremath { \@@_old_ldots } } }
4932 \bool_gset_true:N \g_@@_empty_cell_bool
4933 }

4934 \cs_new_protected:Npn \@@_Cdots
4935 { \@@_collect_options:n { \@@_Cdots_i } }
4936 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4937 {
4938     \int_if_zero:nTF \c@jCol
4939     { \@@_error:nn { in~first~col } \Cdots }
4940     {
4941         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4942         { \@@_error:nn { in~last~col } \Cdots }
4943         {
4944             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4945             { #1 , down = #2 , up = #3 , middle = #4 }
4946         }
4947     }
4948     \bool_if:NF \l_@@_nullify_dots_bool
4949     { \phantom { \ensuremath { \@@_old_cdots } } }
4950     \bool_gset_true:N \g_@@_empty_cell_bool
4951 }

4952 \cs_new_protected:Npn \@@_Vdots
4953 { \@@_collect_options:n { \@@_Vdots_i } }
4954 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4955 {
4956     \int_if_zero:nTF \c@iRow
4957     { \@@_error:nn { in~first~row } \Vdots }
4958     {
4959         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4960         { \@@_error:nn { in~last~row } \Vdots }
4961         {
4962             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4963             { #1 , down = #2 , up = #3 , middle = #4 }
4964         }
4965     }
4966     \bool_if:NF \l_@@_nullify_dots_bool
4967     { \phantom { \ensuremath { \@@_old_vdots } } }
4968     \bool_gset_true:N \g_@@_empty_cell_bool
4969 }

4970 \cs_new_protected:Npn \@@_Ddots
4971 { \@@_collect_options:n { \@@_Ddots_i } }
4972 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4973 {
4974     \int_case:nnF \c@iRow
4975     {
4976         0 { \@@_error:nn { in~first~row } \Ddots }
4977         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4978     }
4979     {
4980         \int_case:nnF \c@jCol

```



```

4981     {
4982         0 { \@@_error:nn { in~first~col } \Ddots }
4983         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4984     }
4985     {
4986         \keys_set_known:nn { nicematrix / Ddots } { #1 }
4987         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4988         { #1 , down = #2 , up = #3 , middle = #4 }
4989     }
4990
4991 }
4992 \bool_if:NF \l_@@_nullify_dots_bool
4993 { \phantom { \ensuremath { \@@_old_ddots } } }
4994 \bool_gset_true:N \g_@@_empty_cell_bool
4995 }

4996 \cs_new_protected:Npn \@@_Iddots
4997 { \@@_collect_options:n { \@@_Iddots_i } }
4998 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4999 {
5000     \int_case:nnF \c@iRow
5001     {
5002         0 { \@@_error:nn { in~first~row } \Iddots }
5003         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5004     }
5005     {
5006         \int_case:nnF \c@jCol
5007         {
5008             0 { \@@_error:nn { in~first~col } \Iddots }
5009             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5010         }
5011         {
5012             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5013             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5014             { #1 , down = #2 , up = #3 , middle = #4 }
5015         }
5016     }
5017     \bool_if:NF \l_@@_nullify_dots_bool
5018     { \phantom { \ensuremath { \@@_old_iddots } } }
5019     \bool_gset_true:N \g_@@_empty_cell_bool
5020 }
5021 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5022 \keys_define:nn { nicematrix / Ddots }
5023 {
5024     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5025     draw-first .default:n = true ,
5026     draw-first .value_forbidden:n = true
5027 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5028 \cs_new_protected:Npn \@@_Hspace:
5029 {
5030     \bool_gset_true:N \g_@@_empty_cell_bool
5031     \hspace
5032 }

```

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.

```

5033 \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`.

```

5034 \cs_new:Npn \@@_Hdotsfor:
5035 {
5036   \bool_lazy_and:nnTF
5037     { \int_if_zero_p:n \c@jCol }
5038     { \int_if_zero_p:n \l_@@_first_col_int }
5039     {
5040       \bool_if:NTF \g_@@_after_col_zero_bool
5041       {
5042         \multicolumn { 1 } { c } { }
5043         \@@_Hdotsfor_i
5044       }
5045       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5046     }
5047   {
5048     \multicolumn { 1 } { c } { }
5049     \@@_Hdotsfor_i
5050   }
5051 }

```

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:`).

```

5052 \hook_gput_code:nnn { begindocument } { . }
5053 {
5054   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5055   \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.

```

5056   \cs_new_protected:Npn \@@_Hdotsfor_i
5057     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5058   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5059     {
5060       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5061       {
5062         \@@_Hdotsfor:nnnn
5063         { \int_use:N \c@iRow }
5064         { \int_use:N \c@jCol }
5065         { #2 }
5066         {
5067           #1 , #3 ,
5068           down = \exp_not:n { #4 } ,
5069           up = \exp_not:n { #5 } ,
5070           middle = \exp_not:n { #6 }
5071         }
5072       }
5073       \prg_replicate:nn { #2 - 1 }
5074       {
5075         &
5076         \multicolumn { 1 } { c } { }
5077         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5078       }
5079     }
5080 }

```

```

5081 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5082 {
5083   \bool_set_false:N \l_@@_initial_open_bool
5084   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```
5085 \int_set:Nn \l_@@_initial_i_int { #1 }
5086 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

For the column, it's a bit more complicated.

```
5087 \int_compare:nNnTF { #2 } = \c_one_int
5088 {
5089   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5090   \bool_set_true:N \l_@@_initial_open_bool
5091 }
5092 {
5093   \cs_if_exist:cTF
5094   {
5095     pgf @ sh @ ns @ \@@_env:
5096     - \int_use:N \l_@@_initial_i_int
5097     - \int_eval:n { #2 - 1 }
5098   }
5099   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5100   {
5101     \int_set:Nn \l_@@_initial_j_int { #2 }
5102     \bool_set_true:N \l_@@_initial_open_bool
5103   }
5104 }
5105 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5106 {
5107   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5108   \bool_set_true:N \l_@@_final_open_bool
5109 }
5110 {
5111   \cs_if_exist:cTF
5112   {
5113     pgf @ sh @ ns @ \@@_env:
5114     - \int_use:N \l_@@_final_i_int
5115     - \int_eval:n { #2 + #3 }
5116   }
5117   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5118   {
5119     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5120     \bool_set_true:N \l_@@_final_open_bool
5121   }
5122 }
5123 \group_begin:
5124 \@@_open_shorten:
5125 \int_if_zero:nTF { #1 }
5126 { \color { nicematrix-first-row } }
5127 {
5128   \int_compare:nNnT { #1 } = \g_@@_row_total_int
5129   { \color { nicematrix-last-row } }
5130 }
5131
5132 \keys_set:nn { nicematrix / xdots } { #4 }
5133 \@@_color:o \l_@@_xdots_color_tl
5134 \@@_actually_draw_ldots:
5135 \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).

```
5136 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5137 { \cs_set_nopar:cpn { @@ _ dotted _ #1 - ##1 } { } }
5138 }
```

```

5139 \hook_gput_code:nnn { begindocument } { . }
5140 {
5141   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5142   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5143   \cs_new_protected:Npn \@@_Vdotsfor:
5144     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5145   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5146     {
5147       \bool_gset_true:N \g_@@_empty_cell_bool
5148       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5149         {
5150           \@@_Vdotsfor:nnnn
5151             { \int_use:N \c@iRow }
5152             { \int_use:N \c@jCol }
5153             { #2 }
5154             {
5155               #1 , #3 ,
5156               down = \exp_not:n { #4 } ,
5157               up = \exp_not:n { #5 } ,
5158               middle = \exp_not:n { #6 }
5159             }
5160           }
5161         }
5162     }

5163 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5164 {
5165   \bool_set_false:N \l_@@_initial_open_bool
5166   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5167   \int_set:Nn \l_@@_initial_j_int { #2 }
5168   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5169   \int_compare:nNnTF { #1 } = \c_one_int
5170   {
5171     \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5172     \bool_set_true:N \l_@@_initial_open_bool
5173   }
5174   {
5175     \cs_if_exist:cTF
5176     {
5177       pgf @ sh @ ns @ \@@_env:
5178       - \int_eval:n { #1 - 1 }
5179       - \int_use:N \l_@@_initial_j_int
5180     }
5181     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5182     {
5183       \int_set:Nn \l_@@_initial_i_int { #1 }
5184       \bool_set_true:N \l_@@_initial_open_bool
5185     }
5186   }
5187   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5188   {
5189     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5190     \bool_set_true:N \l_@@_final_open_bool
5191   }
5192   {
5193     \cs_if_exist:cTF
5194     {
5195       pgf @ sh @ ns @ \@@_env:
5196       - \int_eval:n { #1 + #3 }

```

```

5197         - \int_use:N \l_@@_final_j_int
5198     }
5199     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5200     {
5201         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5202         \bool_set_true:N \l_@@_final_open_bool
5203     }
5204 }
5205 \group_begin:
5206 \@@_open_shorten:
5207 \int_if_zero:nTF { #2 }
5208 { \color { nicematrix-first-col } }
5209 {
5210     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5211     { \color { nicematrix-last-col } }
5212 }
5213 \keys_set:nn { nicematrix / xdots } { #4 }
5214 \@@_color:o \l_@@_xdots_color_tl
5215 \@@_actually_draw_Vdots:
5216 \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).

```

5217     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5218     { \cs_set_nopar:cpn { @@ _ dotted _ ##1 - #2 } { } }
5219 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5220 \NewDocumentCommand \@@_rotate: { 0 { } }
5221 {
5222     \peek_remove_spaces:n
5223     {
5224         \bool_gset_true:N \g_@@_rotate_bool
5225         \keys_set:nn { nicematrix / rotate } { #1 }
5226     }
5227 }
5228 \keys_define:nn { nicematrix / rotate }
5229 {
5230     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5231     c .value_forbidden:n = true ,
5232     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5233 }

```

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

```

5234 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5235 {
5236   \tl_if_empty:nTF { #2 }
5237   { #1 }
5238   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5239 }
5240 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5241 { \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).

```

5242 \hook_gput_code:nnn { begindocument } { . }
5243 {
5244   \cs_set_nopar:Npn \l_@@_argspec_tl
5245   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5246   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5247   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5248   {
5249     \group_begin:
5250     \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5251     \@@_color:o \l_@@_xdots_color_tl
5252     \use:e
5253     {
5254       \@@_line_i:nn
5255       { \@@_double_int_eval:n #2 - \q_stop }
5256       { \@@_double_int_eval:n #3 - \q_stop }
5257     }
5258     \group_end:
5259   }
5260 }
5261 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5262 {
5263   \bool_set_false:N \l_@@_initial_open_bool
5264   \bool_set_false:N \l_@@_final_open_bool
5265   \bool_lazy_or:nnTF
5266   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5267   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5268   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5269   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5270 }
5271 \hook_gput_code:nnn { begindocument } { . }
5272 {
5273   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5274   {
```

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:`.

```

5275   \c_@@_pgfortikzpicture_tl
5276   \@@_draw_line_iii:nn { #1 } { #2 }
5277   \c_@@_endpgfortikzpicture_tl
5278 }
5279 }
```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

5280 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5281 {
5282   \pgfrememberpicturepositiononpagetrue
5283   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5284   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5285   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5286   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5287   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5288   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5289   \@@_draw_line:
5290 }

```

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`

```

5291 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5292 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5293 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5294 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5295 {
5296   \tl_gput_right:Ne \g_@@_row_style_tl
5297   {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5298   \exp_not:N
5299   \@@_if_row_less_than:nn
5300   { \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`).

```

5301   { \exp_not:n { #1 } \scan_stop: }
5302 }
5303 }

```

```

5304 \keys_define:nn { nicematrix / RowStyle }
5305 {
5306   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5307   cell-space-top-limit .value_required:n = true ,
5308   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5309   cell-space-bottom-limit .value_required:n = true ,
5310   cell-space-limits .meta:n =
5311   {

```

```

5312     cell-space-top-limit = #1 ,
5313     cell-space-bottom-limit = #1 ,
5314   } ,
5315   color .tl_set:N = \l_@@_color_tl ,
5316   color .value_required:n = true ,
5317   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5318   bold .default:n = true ,
5319   nb-rows .code:n =
5320     \str_if_eq:eeTF { #1 } { * }
5321     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5322     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5323   nb-rows .value_required:n = true ,
5324   fill .tl_set:N = \l_@@_fill_tl ,
5325   fill .value_required:n = true ,
5326   opacity .tl_set:N = \l_@@_opacity_tl ,
5327   opacity .value_required:n = true ,
5328   rowcolor .tl_set:N = \l_@@_fill_tl ,
5329   rowcolor .value_required:n = true ,
5330   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5331 }

```

```

5332 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5333 {
5334   \group_begin:
5335   \tl_clear:N \l_@@_fill_tl
5336   \tl_clear:N \l_@@_opacity_tl
5337   \tl_clear:N \l_@@_color_tl
5338   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5339   \dim_zero:N \l_tmpa_dim
5340   \dim_zero:N \l_tmpb_dim
5341   \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` (of its alias `fill`) has been used.

```

5342   \tl_if_empty:NF \l_@@_fill_tl
5343   {
5344     \@@_add_opacity_to_fill:

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5345     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5346     {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5347         \@@_exp_color_arg:No \@@_rectanglecolor \l_@@_fill_tl
5348         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5349         { \int_use:N \c@iRow - * }
5350     }

```

Then, the other rows (if there is several rows).

```

5351     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5352     {
5353       \tl_gput_right:Ne \g_@@_pre_code_before_tl
5354       {
5355         \@@_exp_color_arg:No \@@_rowcolor \l_@@_fill_tl
5356         {
5357           \int_eval:n { \c@iRow + 1 }
5358           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5359         }
5360       }
5361     }
5362   }
5363   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```


`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5364   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5365   {
5366     \@@_put_in_row_style:e
5367     {
5368       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5369       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5370         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5371         { \dim_use:N \l_tmpa_dim }
5372     }
5373 }
5374 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5375   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5376   {
5377     \@@_put_in_row_style:e
5378     {
5379       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5380       {
5381         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5382         { \dim_use:N \l_tmpb_dim }
5383       }
5384     }
5385   }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5386   \tl_if_empty:NF \l_@@_color_tl
5387   {
5388     \@@_put_in_row_style:e
5389     {
5390       \mode_leave_vertical:
5391       \@@_color:n { \l_@@_color_tl }
5392     }
5393   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5394   \bool_if:NT \l_@@_bold_row_style_bool
5395   {
5396     \@@_put_in_row_style:n
5397     {
5398       \exp_not:n
5399       {
5400         \if_mode_math:
5401           \c_math_toggle_token
5402           \bfseries \boldmath
5403           \c_math_toggle_token
5404         \else:
5405           \bfseries \boldmath
5406         \fi:
5407       }
5408     }
5409   }
5410   \group_end:
5411   \g_@@_row_style_tl
5412   \ignorespaces
5413 }

```

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).

```
5414 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5415 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5416 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5417 {
```

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.

```
5418 \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`.

```
5419 \str_if_in:nnF { #1 } { !! }
5420 {
5421 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5422 { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5423 }
5424 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5425 {
5426 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5427 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5428 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5429 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5430 }
```

The following command must be used within a `\pgfpicture`.

```
5431 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5432 {
5433 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5434 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5435     \group_begin:
5436     \pgfsetcornersarced
5437     {
5438         \pgfpoint
5439         { \l_@@_tab_rounded_corners_dim }
5440         { \l_@@_tab_rounded_corners_dim }
5441     }

```

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.

```

5442     \bool_if:NTF \l_@@_hvlines_bool
5443     {
5444         \pgfpathrectanglecorners
5445         {
5446             \pgfpointadd
5447             { \@@_qpoint:n { row-1 } }
5448             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5449         }
5450         {
5451             \pgfpointadd
5452             {
5453                 \@@_qpoint:n
5454                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5455             }
5456             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5457         }
5458     }
5459     {
5460         \pgfpathrectanglecorners
5461         { \@@_qpoint:n { row-1 } }
5462         {
5463             \pgfpointadd
5464             {
5465                 \@@_qpoint:n
5466                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5467             }
5468             { \pgfpoint \c_zero_dim \arrayrulewidth }
5469         }
5470     }
5471     \pgfusepath { clip }
5472     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5473     }
5474 }

```

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`).

```

5475 \cs_new_protected:Npn \@@_actually_color:
5476 {
5477     \pgfpicture
5478     \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.

```

5479     \@@_clip_with_rounded_corners:
5480     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5481     {
5482         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5483     {
5484         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5485         \use:c { g_@@_color _ 1 _tl }
5486         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5487     }
5488     {
5489         \begin { pgfscope }
5490             \@@_color_opacity ##2
5491             \use:c { g_@@_color _ ##1 _tl }
5492             \tl_gclear:c { g_@@_color _ ##1 _tl }
5493             \pgfusepath { fill }
5494         \end { pgfscope }
5495     }
5496 }
5497 \endpgfpicture
5498 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5499 \cs_new_protected:Npn \@@_color_opacity
5500 {
5501     \peek_meaning:NTF [
5502         { \@@_color_opacity:w }
5503         { \@@_color_opacity:w [ ] }
5504     }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by currying.

```

5505 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5506 {
5507     \tl_clear:N \l_tmpa_tl
5508     \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.

```

5509     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5510     \tl_if_empty:NTF \l_tmpb_tl
5511         { \@declaredcolor }
5512         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5513 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5514 \keys_define:nn { nicematrix / color-opacity }
5515 {
5516     opacity .tl_set:N          = \l_tmpa_tl ,
5517     opacity .value_required:n = true
5518 }

5519 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5520 {
5521     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5522     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5523     \@@_cartesian_path:
5524 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5525 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5526 {
5527     \tl_if_blank:nF { #2 }
5528     {

```

```

5529 \@@_add_to_colors_seq:en
5530 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5531 { \@@_cartesian_color:nn { #3 } { - } }
5532 }
5533 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5534 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5535 {
5536   \tl_if_blank:nF { #2 }
5537   {
5538     \@@_add_to_colors_seq:en
5539     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5540     { \@@_cartesian_color:nn { - } { #3 } }
5541   }
5542 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5543 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5544 {
5545   \tl_if_blank:nF { #2 }
5546   {
5547     \@@_add_to_colors_seq:en
5548     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5549     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5550   }
5551 }

```

The last argument is the radius of the corners of the rectangle.

```

5552 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5553 {
5554   \tl_if_blank:nF { #2 }
5555   {
5556     \@@_add_to_colors_seq:en
5557     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5558     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5559   }
5560 }

```

The last argument is the radius of the corners of the rectangle.

```

5561 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5562 {
5563   \@@_cut_on_hyphen:w #1 \q_stop
5564   \tl_clear_new:N \l_@@_tmpc_tl
5565   \tl_clear_new:N \l_@@_tmpd_tl
5566   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5567   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5568   \@@_cut_on_hyphen:w #2 \q_stop
5569   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5570   \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`.

```

5571 \@@_cartesian_path:n { #3 }
5572 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5573 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5574 {
5575   \clist_map_inline:nn { #3 }
5576   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5577 }

```

```

5578 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5579 {
5580   \int_step_inline:nn \c@iRow
5581   {
5582     \int_step_inline:nn \c@jCol
5583     {
5584       \int_if_even:nTF { #####1 + ##1 }
5585       { \@@_cellcolor [ #1 ] { #2 } }
5586       { \@@_cellcolor [ #1 ] { #3 } }
5587       { ##1 - #####1 }
5588     }
5589   }
5590 }

```

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”.

```

5591 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5592 {
5593   \@@_rectanglecolor [ #1 ] { #2 }
5594   { 1 - 1 }
5595   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5596 }

```

```

5597 \keys_define:nn { nicematrix / rowcolors }
5598 {
5599   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5600   respect-blocks .default:n = true ,
5601   cols .tl_set:N = \l_@@_cols_tl ,
5602   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5603   restart .default:n = true ,
5604   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5605 }

```

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*.

```

5606 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5607 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5608   \group_begin:
5609   \seq_clear_new:N \l_@@_colors_seq
5610   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5611   \tl_clear_new:N \l_@@_cols_tl
5612   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5613   \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).

```

5614   \int_zero_new:N \l_@@_color_int
5615   \int_set_eq:NN \l_@@_color_int \c_one_int
5616   \bool_if:NT \l_@@_respect_blocks_bool
5617   {

```

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`).

```

5618     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5619     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5620     { \@@_not_in_exterior_p:nnnnn ##1 }
5621   }
5622   \pgfpicture
5623   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5624   \clist_map_inline:nn { #2 }
5625   {
5626     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5627     \tl_if_in:NnTF \l_tmpa_tl { - }
5628     { \@@_cut_on_hyphen:w ##1 \q_stop }
5629     { \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.

```

5630     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5631     \int_set:Nn \l_@@_color_int
5632     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5633     \int_zero_new:N \l_@@_tmpc_int
5634     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5635     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5636     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5637         \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).

```

5638         \bool_if:NT \l_@@_respect_blocks_bool
5639         {
5640           \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5641           { \@@_intersect_our_row_p:nnnnn #####1 }
5642           \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5643         }
5644         \tl_set:No \l_@@_rows_tl
5645         { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5646         \tl_clear_new:N \l_@@_color_tl
5647         \tl_set:Ne \l_@@_color_tl
5648         {
5649           \@@_color_index:n
5650           {
5651             \int_mod:nn
5652             { \l_@@_color_int - 1 }
5653             { \seq_count:N \l_@@_colors_seq }
5654             + 1
5655           }
5656         }
5657         \tl_if_empty:NF \l_@@_color_tl
5658         {
5659           \@@_add_to_colors_seq:ee
5660           { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5661           { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5662         }
5663         \int_incr:N \l_@@_color_int
5664         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5665       }
5666     }
5667   \endpgfpicture

```

```

5668 \group_end:
5669 }

```

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.

```

5670 \cs_new:Npn \@@_color_index:n #1
5671 {

```

Be careful: this command `\@@_color_index:n` must be “*fully expandable*”.

```

5672 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5673 { \@@_color_index:n { #1 - 1 } }
5674 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5675 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5676 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5677 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5678 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5679 {
5680 \int_compare:nNnT { #3 } > \l_tmpb_int
5681 { \int_set:Nn \l_tmpb_int { #3 } }
5682 }

```

```

5683 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5684 {
5685 \int_if_zero:nTF { #4 }
5686 \prg_return_false:
5687 {
5688 \int_compare:nNnTF { #2 } > \c_jCol
5689 \prg_return_false:
5690 \prg_return_true:
5691 }
5692 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5693 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5694 {
5695 \int_compare:nNnTF { #1 } > \l_tmpa_int
5696 \prg_return_false:
5697 {
5698 \int_compare:nNnTF \l_tmpa_int > { #3 }
5699 \prg_return_false:
5700 \prg_return_true:
5701 }
5702 }

```

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`).

```

5703 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5704 {
5705 \dim_compare:nNnTF { #1 } = \c_zero_dim

```



```

5706 {
5707   \bool_if:NTF
5708     \l_@@_nocolor_used_bool
5709     \@@_cartesian_path_normal_ii:
5710   {
5711     \clist_if_empty:NTF \l_@@_corners_cells_clist
5712       { \@@_cartesian_path_normal_i:n { #1 } }
5713       \@@_cartesian_path_normal_ii:
5714   }
5715 }
5716 { \@@_cartesian_path_normal_i:n { #1 } }
5717 }

```

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.

```

5718 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5719 {
5720   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5721   \clist_map_inline:Nn \l_@@_cols_tl
5722   {
5723     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5724     \tl_if_in:NnTF \l_tmpa_tl { - }
5725       { \@@_cut_on_hyphen:w ##1 \q_stop }
5726       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5727     \tl_if_empty:NTF \l_tmpa_tl
5728       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5729       {
5730         \str_if_eq:eeT \l_tmpa_tl { * }
5731         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5732       }
5733     \tl_if_empty:NTF \l_tmpb_tl
5734       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5735       {
5736         \str_if_eq:eeT \l_tmpb_tl { * }
5737         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5738       }
5739     \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5740       { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

\l_@@_tmpc_tl will contain the number of column.

```

5741     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5742     \@@_qpoint:n { col - \l_tmpa_tl }
5743     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5744       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5745       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5746     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5747     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5748   \clist_map_inline:Nn \l_@@_rows_tl
5749   {
5750     \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5751     \tl_if_in:NnTF \l_tmpa_tl { - }
5752       { \@@_cut_on_hyphen:w ####1 \q_stop }
5753       { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5754     \tl_if_empty:NTF \l_tmpa_tl
5755       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5756       {
5757         \str_if_eq:eeT \l_tmpa_tl { * }
5758         { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5759       }
5760     \tl_if_empty:NTF \l_tmpb_tl

```

```

5761         { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5762         {
5763             \str_if_eq:eeT \l_tmpb_tl { * }
5764             { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5765         }
5766         \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5767         { \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`.

```

5768         \cs_if_exist:cF
5769         { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
5770         {
5771             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5772             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5773             \@@_qpoint:n { row - \l_tmpa_tl }
5774             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5775             \pgfpathrectanglecorners
5776             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5777             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5778         }
5779     }
5780 }
5781 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5782 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5783 {
5784     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5785     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5786     \clist_map_inline:Nn \l_@@_cols_tl
5787     {
5788         \@@_qpoint:n { col - ##1 }
5789         \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5790         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5791         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5792         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5793         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5794     \clist_map_inline:Nn \l_@@_rows_tl
5795     {
5796         \@@_if_in_corner:nF { #####1 - ##1 }
5797         {
5798             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5799             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5800             \@@_qpoint:n { row - #####1 }
5801             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5802             \cs_if_exist:cF { @@ _ nocolor _ #####1 - ##1 }
5803             {
5804                 \pgfpathrectanglecorners
5805                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5806                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5807             }
5808         }
5809     }
5810 }
5811 }

```

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`).

```

5812 \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

```

5813 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5814 {
5815   \bool_set_true:N \l_@@_nocolor_used_bool
5816   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5817   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5818   \clist_map_inline:Nn \l_@@_rows_tl
5819   {
5820     \clist_map_inline:Nn \l_@@_cols_tl
5821     { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
5822   }
5823 }

```

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`.

```

5824 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5825 {
5826   \clist_set_eq:NN \l_tmpa_clist #1
5827   \clist_clear:N #1
5828   \clist_map_inline:Nn \l_tmpa_clist
5829   {
5830     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5831     \tl_if_in:NnTF \l_tmpa_tl { - }
5832     { \@@_cut_on_hyphen:w ##1 \q_stop }
5833     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5834     \bool_lazy_or:nnT
5835     { \str_if_eq_p:ee \l_tmpa_tl { * } }
5836     { \tl_if_blank_p:o \l_tmpa_tl }
5837     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5838     \bool_lazy_or:nnT
5839     { \str_if_eq_p:ee \l_tmpb_tl { * } }
5840     { \tl_if_blank_p:o \l_tmpb_tl }
5841     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5842     \int_compare:nNnT \l_tmpb_tl > #2
5843     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5844     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5845     { \clist_put_right:Nn #1 { ####1 } }
5846   }
5847 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5848 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5849 {
5850   \@@_test_color_inside:
5851   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5852   {

```

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`).

```

5853     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5854     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5855   }
5856   \ignorespaces
5857 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5858 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5859 {
5860   \@@_test_color_inside:
5861   \tl_gput_right:N \g_@@_pre_code_before_tl
5862   {
5863     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5864     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5865     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5866   }
5867   \ignorespaces
5868 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5869 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5870 { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }

```

The braces around `#2` and `#3` are mandatory.

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5871 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5872 {
5873   \@@_test_color_inside:
5874   \peek_remove_spaces:n
5875   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5876 }

5877 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5878 {

```

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`.

```

5879 \seq_gclear:N \g_tmpa_seq
5880 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5881 { \@@_rowlistcolors_tabular_i:nnnn #1 }
5882 \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`.

```

5883 \seq_gput_right:N \g_@@_rowlistcolors_seq
5884 {
5885   { \int_use:N \c@iRow }
5886   { \exp_not:n { #1 } }
5887   { \exp_not:n { #2 } }
5888   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5889 }
5890 }

```

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`).

```
5891 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5892 {
5893   \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).

```
5894   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5895   {
5896     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5897     {
5898       \@@_rowlistcolors
5899       [ \exp_not:n { #2 } ]
5900       { #1 - \int_eval:n { \c@iRow - 1 } }
5901       { \exp_not:n { #3 } }
5902       [ \exp_not:n { #4 } ]
5903     }
5904   }
5905 }
```

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.

```
5906 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5907 {
5908   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5909   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5910   \seq_gclear:N \g_@@_rowlistcolors_seq
5911 }

5912 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5913 {
5914   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5915   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5916 }
```

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.

```
5917 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5918 {
```

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).

```
5919   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5920   {
```

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).

```
5921   \tl_gput_left:Ne \g_@@_pre_code_before_tl
5922   {
5923     \exp_not:N \columncolor [ #1 ]
5924     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5925   }
5926 }
5927 }
```

```

5928 \hook_gput_code:nnn { begindocument } { . }
5929 {
5930   \IfPackageLoadedTF { colortbl }
5931   {
5932     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5933     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5934     \cs_new_protected:Npn \@@_revert_colortbl:
5935     {
5936       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5937       {
5938         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5939         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5940       }
5941     }
5942   }
5943   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5944 }

```

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`.

```

5945 \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`.

```

5946 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5947 {
5948   \int_if_zero:nTF \l_@@_first_col_int
5949   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5950   {
5951     \int_if_zero:nTF \c@jCol
5952     {
5953       \int_compare:nNf \c@iRow = { -1 }
5954       { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5955     }
5956     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5957   }
5958 }

```

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.

```

5959 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5960 {
5961   \int_if_zero:nF \c@iRow
5962   {
5963     \int_compare:nNf \c@iRow = \l_@@_last_row_int
5964     {

```

```

5965         \int_compare:nNnT \c@jCol > \c_zero_int
5966         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5967     }
5968 }
5969 }

```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to `-2` or `-1` (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```

5970 \keys_define:nn { nicematrix / Rules }
5971 {
5972     position .int_set:N = \l_@@_position_int ,
5973     position .value_required:n = true ,
5974     start .int_set:N = \l_@@_start_int ,
5975     end .code:n =
5976         \bool_lazy_or:nnTF
5977         { \tl_if_empty_p:n { #1 } }
5978         { \str_if_eq_p:ee { #1 } { last } }
5979         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5980         { \int_set:Nn \l_@@_end_int { #1 } }
5981 }

```

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.

```

5982 \keys_define:nn { nicematrix / RulesBis }
5983 {
5984     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5985     multiplicity .initial:n = 1 ,
5986     dotted .bool_set:N = \l_@@_dotted_bool ,
5987     dotted .initial:n = false ,
5988     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]`.

```

5989     color .code:n =
5990         \@@_set_CT@arc@:n { #1 }
5991         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5992     color .value_required:n = true ,
5993     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5994     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.

```

5995     tikz .code:n =
5996         \IfPackageLoadedTF { tikz }
5997         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5998         { \@@_error:n { tikz~without~tikz } } ,
5999     tikz .value_required:n = true ,
6000     total-width .dim_set:N = \l_@@_rule_width_dim ,

```

```

6001     total-width .value_required:n = true ,
6002     width .meta:n = { total-width = #1 } ,
6003     unknown .code:n = \@_error:n { Unknow~key~for~RulesBis }
6004 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6005 \cs_new_protected:Npn \@_vline:n #1
6006 {

```

The group is for the options.

```

6007     \group_begin:
6008     \int_set_eq:NN \l_@@_end_int \c@iRow
6009     \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

6010     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6011     \@_vline_i:
6012     \group_end:
6013 }

```

```

6014 \cs_new_protected:Npn \@_vline_i:
6015 {

```

`\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`.

```

6016     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6017     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6018     \l_tmpa_tl
6019     {

```

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.

```

6020         \bool_gset_true:N \g_tmpa_bool
6021         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6022             { \@_test_vline_in_block:nnnnn ##1 }
6023         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6024             { \@_test_vline_in_block:nnnnn ##1 }
6025         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6026             { \@_test_vline_in_stroken_block:nnnn ##1 }
6027         \clist_if_empty:NF \l_@@_corners_clist \@_test_in_corner_v:
6028         \bool_if:NTF \g_tmpa_bool
6029             {
6030                 \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.

```

6031             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6032         }
6033     {
6034         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6035         {
6036             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6037             \@_vline_ii:
6038             \int_zero:N \l_@@_local_start_int
6039         }
6040     }
6041 }
6042 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int

```



```

6043     {
6044         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6045         \@@_vline_ii:
6046     }
6047 }

6048 \cs_new_protected:Npn \@@_test_in_corner_v:
6049 {
6050     \int_compare:nNnTF \l_tmpb_tl = { \c@jCol + 1 }
6051     {
6052         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6053         { \bool_set_false:N \g_tmpa_bool }
6054     }
6055     {
6056         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6057         {
6058             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6059             { \bool_set_false:N \g_tmpa_bool }
6060             {
6061                 \@@_if_in_corner:nT
6062                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6063                 { \bool_set_false:N \g_tmpa_bool }
6064             }
6065         }
6066     }
6067 }

6068 \cs_new_protected:Npn \@@_vline_ii:
6069 {
6070     \tl_clear:N \l_@@_tikz_rule_tl
6071     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6072     \bool_if:NTF \l_@@_dotted_bool
6073     \@@_vline_iv:
6074     {
6075         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6076         \@@_vline_iii:
6077         \@@_vline_v:
6078     }
6079 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6080 \cs_new_protected:Npn \@@_vline_iii:
6081 {
6082     \pgfpicture
6083     \pgfrememberpicturepositiononpagetrue
6084     \pgf@relevantforpicturesizefalse
6085     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6086     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6087     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6088     \dim_set:Nn \l_tmpb_dim
6089     {
6090         \pgf@x
6091         - 0.5 \l_@@_rule_width_dim
6092         +
6093         ( \arrayrulewidth * \l_@@_multiplicity_int
6094           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6095     }
6096     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6097     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6098     \bool_lazy_all:nT
6099     {

```

```

6100     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6101     { \cs_if_exist_p:N \CT@drsc@ }
6102     { ! \tl_if_blank_p:o \CT@drsc@ }
6103   }
6104   {
6105     \group_begin:
6106     \CT@drsc@
6107     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6108     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6109     \dim_set:Nn \l_@@_tmpd_dim
6110       {
6111         \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6112         * ( \l_@@_multiplicity_int - 1 )
6113       }
6114     \pgfpathrectanglecorners
6115       { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6116       { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6117     \pgfusepath { fill }
6118     \group_end:
6119   }
6120   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6121   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6122   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6123   {
6124     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6125     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6126     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6127     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6128   }
6129   \CT@arc@
6130   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6131   \pgfsetrectcap
6132   \pgfusepathqstroke
6133   \endpgfpicture
6134 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6135 \cs_new_protected:Npn \@@_vline_iv:
6136 {
6137   \pgfpicture
6138   \pgfrememberpicturepositiononpagetrue
6139   \pgf@relevantforpicturesizefalse
6140   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6141   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6142   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6143   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6144   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6145   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6146   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6147   \CT@arc@
6148   \@@_draw_line:
6149   \endpgfpicture
6150 }

```

The following code is for the case when the user uses the key `tikz`.

```

6151 \cs_new_protected:Npn \@@_vline_v:
6152 {
6153   \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).

```

6154 \CT@arc@
6155 \tl_if_empty:NF \l_@@_rule_color_tl
6156 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6157 \pgfrememberpicturepositiononpagetrue
6158 \pgf@relevantforpicturesizefalse
6159 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6160 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6161 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6162 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6163 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6164 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6165 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6166 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6167 ( \l_tmpb_dim , \l_tmpa_dim ) --
6168 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6169 \end { tikzpicture }
6170 }

```

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).

```

6171 \cs_new_protected:Npn \@@_draw_vlines:
6172 {
6173   \int_step_inline:nnn
6174   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6175   {
6176     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6177     \c@jCol
6178     { \int_eval:n { \c@jCol + 1 } } }
6179   }
6180   {
6181     \str_if_eq:eeF \l_@@_vlines_clist { all }
6182     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6183     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6184   }
6185 }

```

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}`.

```

6186 \cs_new_protected:Npn \@@_hline:n #1
6187 {

```

The group is for the options.

```

6188   \group_begin:
6189   \int_zero_new:N \l_@@_end_int
6190   \int_set_eq:NN \l_@@_end_int \c@jCol
6191   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6192   \@@_hline_i:
6193   \group_end:
6194 }

6195 \cs_new_protected:Npn \@@_hline_i:
6196 {
6197   \int_zero_new:N \l_@@_local_start_int
6198   \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`.

```

6199   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6200   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6201   \l_tmpb_tl
6202   {

```

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.

```
6203         \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
6204     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6205     { \@@_test_hline_in_block:nnnnn ##1 }
6206     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6207     { \@@_test_hline_in_block:nnnnn ##1 }
6208     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6209     { \@@_test_hline_in_stroken_block:nnnn ##1 }
6210     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6211     \bool_if:NTF \g_tmpa_bool
6212     {
6213         \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.

```
6214         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6215     }
6216     {
6217         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6218         {
6219             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6220             \@@_hline_ii:
6221             \int_zero:N \l_@@_local_start_int
6222         }
6223     }
6224 }
6225 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6226 {
6227     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6228     \@@_hline_ii:
6229 }
6230 }
```

```
6231 \cs_new_protected:Npn \@@_test_in_corner_h:
6232 {
6233     \int_compare:nNnTF \l_tmpa_tl = { \c@iRow + 1 }
6234     {
6235         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6236         { \bool_set_false:N \g_tmpa_bool }
6237     }
6238     {
6239         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
6240         {
6241             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6242             { \bool_set_false:N \g_tmpa_bool }
6243             {
6244                 \@@_if_in_corner:nT
6245                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6246                 { \bool_set_false:N \g_tmpa_bool }
6247             }
6248         }
6249     }
6250 }
```

```
6251 \cs_new_protected:Npn \@@_hline_ii:
6252 {
```

```

6253 \tl_clear:N \l_@@_tikz_rule_tl
6254 \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6255 \bool_if:NTF \l_@@_dotted_bool
6256 \@@_hline_iv:
6257 {
6258   \tl_if_empty:NTF \l_@@_tikz_rule_tl
6259     \@@_hline_iii:
6260     \@@_hline_v:
6261 }
6262 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6263 \cs_new_protected:Npn \@@_hline_iii:
6264 {
6265   \pgfpicture
6266   \pgfrememberpicturepositiononpagetrue
6267   \pgf@relevantforpicturesizefalse
6268   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6269   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6270   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6271   \dim_set:Nn \l_tmpb_dim
6272   {
6273     \pgf@y
6274     - 0.5 \l_@@_rule_width_dim
6275     +
6276     ( \arrayrulewidth * \l_@@_multiplicity_int
6277       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6278   }
6279   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6280   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6281   \bool_lazy_all:nT
6282   {
6283     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6284     { \cs_if_exist_p:N \CT@drsc@ }
6285     { ! \tl_if_blank_p:o \CT@drsc@ }
6286   }
6287   {
6288     \group_begin:
6289     \CT@drsc@
6290     \dim_set:Nn \l_@@_tmpd_dim
6291     {
6292       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6293       * ( \l_@@_multiplicity_int - 1 )
6294     }
6295     \pgfpathrectanglecorners
6296     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6297     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6298     \pgfusepathqfill
6299     \group_end:
6300   }
6301   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6302   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6303   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6304   {
6305     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6306     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6307     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6308     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6309   }
6310   \CT@arc@
6311   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6312   \pgfsetrectcap
6313   \pgfusepathqstroke

```

```

6314 \endpgfpicture
6315 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6316 \cs_new_protected:Npn \@@_hline_iv:
6317 {
6318 \pgfpicture
6319 \pgfrememberpicturepositiononpagetrue
6320 \pgf@relevantforpicturesizefalse
6321 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6322 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6323 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6324 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6325 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6326 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6327 {
6328 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6329 \bool_if:NF \g_@@_delims_bool
6330 { \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.

```

6331 \tl_if_eq:NnF \g_@@_left_delim_tl (
6332 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6333 )
6334 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6335 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6336 \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6337 {
6338 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6339 \bool_if:NF \g_@@_delims_bool
6340 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6341 \tl_if_eq:NnF \g_@@_right_delim_tl )
6342 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6343 }
6344 \CT@arc@
6345 \@@_draw_line:
6346 \endpgfpicture
6347 }

```

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`).

```

6348 \cs_new_protected:Npn \@@_hline_v:
6349 {
6350 \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).

```

6351 \CT@arc@
6352 \tl_if_empty:NF \l_@@_rule_color_tl
6353 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6354 \pgfrememberpicturepositiononpagetrue
6355 \pgf@relevantforpicturesizefalse
6356 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6357 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6358 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6359 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6360 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6361 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6362 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6363 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6364 ( \l_tmpa_dim , \l_tmpb_dim ) --
6365 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6366 \end { tikzpicture }
6367 }

```

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).

```

6368 \cs_new_protected:Npn \@@_draw_hlines:
6369 {
6370   \int_step_inline:nnn
6371   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6372   {
6373     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6374     \c@iRow
6375     { \int_eval:n { \c@iRow + 1 } }
6376   }
6377   {
6378     \str_if_eq:eeF \l_@@_hlines_clist { all }
6379     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6380     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6381   }
6382 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6383 \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.

```

6384 \cs_set:Npn \@@_Hline_i:n #1
6385 {
6386   \peek_remove_spaces:n
6387   {
6388     \peek_meaning:NTF \Hline
6389     { \@@_Hline_ii:nn { #1 + 1 } }
6390     { \@@_Hline_iii:n { #1 } }
6391   }
6392 }
6393 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6394 \cs_set:Npn \@@_Hline_iii:n #1
6395 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6396 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6397 {
6398   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6399   \skip_vertical:N \l_@@_rule_width_dim
6400   \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

6401     {
6402       \@@_hline:n
6403       {
6404         multiplicity = #1 ,
6405         position = \int_eval:n { \c@iRow + 1 } ,
6406         total-width = \dim_use:N \l_@@_rule_width_dim ,
6407         #2
6408       }
6409     }
6410   \egroup
6411 }

```

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`).

```

6412 \cs_new_protected:Npn \@@_custom_line:n #1
6413 {
6414   \str_clear_new:N \l_@@_command_str
6415   \str_clear_new:N \l_@@_ccommand_str
6416   \str_clear_new:N \l_@@_letter_str
6417   \tl_clear_new:N \l_@@_other_keys_tl
6418   \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6419   \bool_lazy_all:nTF
6420   {
6421     { \str_if_empty_p:N \l_@@_letter_str }
6422     { \str_if_empty_p:N \l_@@_command_str }
6423     { \str_if_empty_p:N \l_@@_ccommand_str }
6424   }
6425   { \@@_error:n { No~letter~and~no~command } }
6426   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6427 }

6428 \keys_define:nn { nicematrix / custom-line }
6429 {
6430   letter .str_set:N = \l_@@_letter_str ,
6431   letter .value_required:n = true ,
6432   command .str_set:N = \l_@@_command_str ,
6433   command .value_required:n = true ,
6434   ccommand .str_set:N = \l_@@_ccommand_str ,
6435   ccommand .value_required:n = true ,
6436 }

```

```

6437 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6438 \cs_new_protected:Npn \@@_custom_line_i:n #1
6439 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6440   \bool_set_false:N \l_@@_tikz_rule_bool
6441   \bool_set_false:N \l_@@_dotted_rule_bool
6442   \bool_set_false:N \l_@@_color_bool

```



```

6443 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6444 \bool_if:NT \l_@@_tikz_rule_bool
6445 {
6446   \IfPackageLoadedF { tikz }
6447   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6448   \bool_if:NT \l_@@_color_bool
6449   { \@@_error:n { color-in-custom-line-with-tikz } }
6450 }
6451 \bool_if:NT \l_@@_dotted_rule_bool
6452 {
6453   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6454   { \@@_error:n { key-multiplicity-with-dotted } }
6455 }
6456 \str_if_empty:NF \l_@@_letter_str
6457 {
6458   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6459   { \@@_error:n { Several-letters } }
6460   {
6461     \tl_if_in:NoTF
6462     \c_@@_forbidden_letters_str
6463     \l_@@_letter_str
6464     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6465   }

```

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.

```

6466 \cs_set_nopar:cpn { @@ _ \l_@@_letter_str } ##1
6467 { \@@_v_custom_line:n { #1 } }
6468 }
6469 }
6470 }
6471 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6472 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6473 }
6474 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6475 \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.

```

6476 \keys_define:nn { nicematrix / custom-line-bis }
6477 {
6478   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6479   multiplicity .initial:n = 1 ,
6480   multiplicity .value_required:n = true ,
6481   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6482   color .value_required:n = true ,
6483   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6484   tikz .value_required:n = true ,
6485   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6486   dotted .value_forbidden:n = true ,
6487   total-width .code:n = { } ,
6488   total-width .value_required:n = true ,
6489   width .code:n = { } ,
6490   width .value_required:n = true ,
6491   sep-color .code:n = { } ,
6492   sep-color .value_required:n = true ,
6493   unknown .code:n = \@@_error:n { Unknown-key-for-custom-line }
6494 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```
6495 \bool_new:N \l_@@_dotted_rule_bool
6496 \bool_new:N \l_@@_tikz_rule_bool
6497 \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`.

```
6498 \keys_define:nn { nicematrix / custom-line-width }
6499 {
6500   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6501   multiplicity .initial:n = 1 ,
6502   multiplicity .value_required:n = true ,
6503   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6504   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6505                       \bool_set_true:N \l_@@_total_width_bool ,
6506   total-width .value_required:n = true ,
6507   width .meta:n = { total-width = #1 } ,
6508   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6509 }
```

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`).

```
6510 \cs_new_protected:Npn \@@_h_custom_line:n #1
6511 {
```

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`).

```
6512   \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6513   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6514 }
```

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`).

```
6515 \cs_new_protected:Npn \@@_c_custom_line:n #1
6516 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6517   \exp_args:Nc \NewExpandableDocumentCommand
6518   { nicematrix - \l_@@_ccommand_str }
6519   { 0 { } m }
6520   {
6521     \noalign
6522     {
6523       \@@_compute_rule_width:n { #1 , ##1 }
6524       \skip_vertical:n { \l_@@_rule_width_dim }
6525       \clist_map_inline:nn
6526       { ##2 }
6527       { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6528     }
6529   }
6530   \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6531 }
```

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*.

```

6532 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6533 {
6534   \tl_if_in:nnTF { #2 } { - }
6535   { \@@_cut_on_hyphen:w #2 \q_stop }
6536   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6537   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6538   {
6539     \@@_hline:n
6540     {
6541       #1 ,
6542       start = \l_tmpa_tl ,
6543       end = \l_tmpb_tl ,
6544       position = \int_eval:n { \c@iRow + 1 } ,
6545       total-width = \dim_use:N \l_@@_rule_width_dim
6546     }
6547   }
6548 }

6549 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6550 {
6551   \bool_set_false:N \l_@@_tikz_rule_bool
6552   \bool_set_false:N \l_@@_total_width_bool
6553   \bool_set_false:N \l_@@_dotted_rule_bool
6554   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6555   \bool_if:NF \l_@@_total_width_bool
6556   {
6557     \bool_if:NTF \l_@@_dotted_rule_bool
6558     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6559     {
6560       \bool_if:NF \l_@@_tikz_rule_bool
6561       {
6562         \dim_set:Nn \l_@@_rule_width_dim
6563         {
6564           \arrayrulewidth * \l_@@_multiplicity_int
6565           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6566         }
6567       }
6568     }
6569   }
6570 }

6571 \cs_new_protected:Npn \@@_v_custom_line:n #1
6572 {
6573   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6574   \tl_gput_right:Ne \g_@@_array_preamble_tl
6575   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6576   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6577   {
6578     \@@_vline:n
6579     {
6580       #1 ,
6581       position = \int_eval:n { \c@jCol + 1 } ,
6582       total-width = \dim_use:N \l_@@_rule_width_dim
6583     }
6584   }
6585   \@@_rec_preamble:n
6586 }

6587 \@@_custom_line:n
6588 { 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.

```

6589 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6590 {
6591   \int_compare:nNnT \l_tmpa_tl > { #1 }
6592   {
6593     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6594     {
6595       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6596       {
6597         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6598         { \bool_gset_false:N \g_tmpa_bool }
6599       }
6600     }
6601   }
6602 }
```

The same for vertical rules.

```

6603 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6604 {
6605   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6606   {
6607     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6608     {
6609       \int_compare:nNnT \l_tmpb_tl > { #2 }
6610       {
6611         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6612         { \bool_gset_false:N \g_tmpa_bool }
6613       }
6614     }
6615   }
6616 }

6617 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6618 {
6619   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6620   {
6621     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6622     {
6623       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6624       { \bool_gset_false:N \g_tmpa_bool }
6625       {
6626         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6627         { \bool_gset_false:N \g_tmpa_bool }
6628       }
6629     }
6630   }
6631 }

6632 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6633 {
6634   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6635   {
6636     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6637     {
6638       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6639       { \bool_gset_false:N \g_tmpa_bool }
6640       {
6641         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6642         { \bool_gset_false:N \g_tmpa_bool }
6643       }
6644     }
6645   }
6646 }
```

```

6644     }
6645   }
6646 }

```

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.

```

6647 \cs_new_protected:Npn \@@_compute_corners:
6648 {
6649   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6650   { \@@_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`).

```

6651   \clist_clear:N \l_@@_corners_cells_clist
6652   \clist_map_inline:Nn \l_@@_corners_clist
6653   {
6654     \str_case:nnF { ##1 }
6655     {
6656       { NW }
6657       { \@@_compute_a_corner:nnnnnn 1 1 1 \c@iRow \c@jCol }
6658       { NE }
6659       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6660       { SW }
6661       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6662       { SE }
6663       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6664     }
6665     { \@@_error:nn { bad~corner } { ##1 } }
6666   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6667   \clist_if_empty:NF \l_@@_corners_cells_clist
6668   {

```

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.

```

6669     \tl_gput_right:Ne \g_@@_aux_tl
6670     {
6671       \cs_set_nopar:Npn \exp_not:N \l_@@_corners_cells_clist
6672       { \l_@@_corners_cells_clist }
6673     }
6674   }
6675 }

```

```

6676 \cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
6677 {
6678   \int_step_inline:nnn { #1 } { #3 }
6679   {
6680     \int_step_inline:nnn { #2 } { #4 }
6681     { \cs_set_nopar:cpn { @@ _ block _ ##1 - #####1 } { } }
6682   }
6683 }

```

```

6684 \prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
6685 {
6686   \cs_if_exist:cTF
6687   { @@_block_ \int_eval:n { #1 } - \int_eval:n { #2 } }
6688   \prg_return_true:
6689   \prg_return_false:
6690 }

```

“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.

```

6691 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6692 {

```

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.

```

6693   \bool_set_false:N \l_tmpa_bool
6694   \int_zero_new:N \l_@@_last_empty_row_int
6695   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6696   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6697   {
6698     \bool_lazy_or:nnTF
6699     {
6700       \cs_if_exist_p:c
6701       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6702     }
6703     { \@@_if_in_block_p:nn { ##1 } { #2 } }
6704     { \bool_set_true:N \l_tmpa_bool }
6705     {
6706       \bool_if:NF \l_tmpa_bool
6707       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6708     }
6709   }

```

Now, you determine the last empty cell in the row of number 1.

```

6710   \bool_set_false:N \l_tmpa_bool
6711   \int_zero_new:N \l_@@_last_empty_column_int
6712   \int_set:Nn \l_@@_last_empty_column_int { #2 }
6713   \int_step_inline:nnnn { #2 } { #4 } { #6 }
6714   {
6715     \bool_lazy_or:nnTF
6716     {
6717       \cs_if_exist_p:c
6718       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6719     }
6720     { \@@_if_in_block_p:nn { #1 } { ##1 } }
6721     { \bool_set_true:N \l_tmpa_bool }
6722     {
6723       \bool_if:NF \l_tmpa_bool
6724       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6725     }
6726   }

```

Now, we loop over the rows.

```
6727 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6728 {
```

We treat the row number ##1 with another loop.

```
6729 \bool_set_false:N \l_tmpa_bool
6730 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6731 {
6732 \bool_lazy_or:nnTF
6733 { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 } }
6734 { \@@_if_in_block_p:nn { ##1 } { #####1 } }
6735 { \bool_set_true:N \l_tmpa_bool }
6736 {
6737 \bool_if:NF \l_tmpa_bool
6738 {
6739 \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6740 \clist_put_right:Nn
6741 \l_@@_corners_cells_clist
6742 { ##1 - #####1 }
6743 \cs_set_nopar:cpn { @@ _ corner _ ##1 - #####1 } { }
6744 }
6745 }
6746 }
6747 }
6748 }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```
6749 \cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
6750 \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:NnT \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.

```
6751 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```
6752 \keys_define:nn { nicematrix / NiceMatrixBlock }
6753 {
6754 auto-columns-width .code:n =
6755 {
6756 \bool_set_true:N \l_@@_block_auto_columns_width_bool
6757 \dim_gzero_new:N \g_@@_max_cell_width_dim
6758 \bool_set_true:N \l_@@_auto_columns_width_bool
6759 }
6760 }
```

```
6761 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6762 {
6763 \int_gincr:N \g_@@_NiceMatrixBlock_int
6764 \dim_zero:N \l_@@_columns_width_dim
6765 \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6766 \bool_if:NT \l_@@_block_auto_columns_width_bool
6767 {
6768 \cs_if_exist:cT
```

```

6769     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6770     {
6771       \dim_set:Nn \l_@@_columns_width_dim
6772       {
6773         \use:c
6774         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6775       }
6776     }
6777   }
6778 }

```

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`).

```

6779 {
6780   \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.

```

6781     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6782     {
6783       \bool_if:NT \l_@@_block_auto_columns_width_bool
6784       {
6785         \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6786         \iow_shipout:Ne \@mainaux
6787         {
6788           \cs_gset:cpn
6789           { @@ _ 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.

```

6790           { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6791         }
6792         \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6793       }
6794     }
6795   \ignorespacesafterend
6796 }

```

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).

```

6797 \cs_new_protected:Npn \@@_create_extra_nodes:
6798 {
6799   \bool_if:nTF \l_@@_medium_nodes_bool
6800   {
6801     \bool_if:NTF \l_@@_large_nodes_bool
6802     \@@_create_medium_and_large_nodes:
6803     \@@_create_medium_nodes:
6804   }
6805   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6806 }

```

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`.

```

6807 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6808 {
6809   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6810   {
6811     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6812     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6813     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6814     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6815   }
6816   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6817   {
6818     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6819     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6820     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6821     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6822   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6823   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6824   {
6825     \int_step_variable:nnNn
6826     \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.

```

6827     {
6828       \cs_if_exist:cT
6829       { 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`.

```

6830     {
6831       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6832       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6833       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6834       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6835       {
6836         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6837         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6838       }

```

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`.

```

6839       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6840       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6841       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6842       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6843       {
6844         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }

```

```

6845         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6846     }
6847 }
6848 }
6849 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6850 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6851 {
6852     \dim_compare:nNnT
6853     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6854     {
6855         \@@_qpoint:n { row - \@@_i: - base }
6856         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6857         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6858     }
6859 }
6860 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6861 {
6862     \dim_compare:nNnT
6863     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6864     {
6865         \@@_qpoint:n { col - \@@_j: }
6866         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6867         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6868     }
6869 }
6870 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6871 \cs_new_protected:Npn \@@_create_medium_nodes:
6872 {
6873     \pgfpicture
6874     \pgfrememberpicturepositiononpagetrue
6875     \pgf@relevantforpicturesizefalse
6876     \@@_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”.

```

6877     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6878     \@@_create_nodes:
6879     \endpgfpicture
6880 }

```

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:`.

```

6881 \cs_new_protected:Npn \@@_create_large_nodes:
6882 {
6883     \pgfpicture
6884     \pgfrememberpicturepositiononpagetrue
6885     \pgf@relevantforpicturesizefalse
6886     \@@_computations_for_medium_nodes:
6887     \@@_computations_for_large_nodes:
6888     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6889     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6890 \endpgfpicture
6891 }
6892 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6893 {
6894 \pgfpicture
6895 \pgfrememberpicturepositiononpagetrue
6896 \pgf@relevantforpicturesizefalse
6897 \@@_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”.

```

6898 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6899 \@@_create_nodes:
6900 \@@_computations_for_large_nodes:
6901 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6902 \@@_create_nodes:
6903 \endpgfpicture
6904 }

```

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.

```

6905 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6906 {
6907 \int_set_eq:NN \l_@@_first_row_int \c_one_int
6908 \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`.

```

6909 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6910 {
6911 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6912 {
6913 (
6914 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6915 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6916 )
6917 / 2
6918 }
6919 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6920 { l_@@_row _ \@@_i: _ min _ dim }
6921 }
6922 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6923 {
6924 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6925 {
6926 (
6927 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6928 \dim_use:c
6929 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6930 )
6931 / 2
6932 }
6933 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6934 { l_@@_column _ \@@_j: _ max _ dim }
6935 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6936 \dim_sub:cn
6937 { l_@@_column _ 1 _ min _ dim }
6938 \l_@@_left_margin_dim
6939 \dim_add:cn
6940 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6941 \l_@@_right_margin_dim
6942 }

```

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`).

```

6943 \cs_new_protected:Npn \@@_create_nodes:
6944 {
6945   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6946   {
6947     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6948     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6949       \@@_pgf_rect_node:nnnnn
6950       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6951       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6952       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6953       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6954       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6955       \str_if_empty:NF \l_@@_name_str
6956       {
6957         \pgfnodealias
6958         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6959         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6960       }
6961     }
6962   }

```

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 .

```

6963   \seq_map_pairwise_function:NNN
6964   \g_@@_multicolumn_cells_seq
6965   \g_@@_multicolumn_sizes_seq
6966   \@@_node_for_multicolumn:nn
6967 }

6968 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6969 {
6970   \cs_set_nopar:Npn \@@_i: { #1 }
6971   \cs_set_nopar:Npn \@@_j: { #2 }
6972 }

```

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”).

```

6973 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6974 {
6975   \@@_extract_coords_values: #1 \q_stop
6976   \@@_pgf_rect_node:nnnnn
6977   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6978   { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6979   { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6980   { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: + #2 - 1 } _max_dim } }
6981   { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6982   \str_if_empty:NF \l_@@_name_str
6983   {
6984     \pgfnodealias
6985     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6986     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6987   }
6988 }

```

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).

```

6989 \keys_define:nn { nicematrix / Block / FirstPass }
6990 {
6991   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
6992     \bool_set_true:N \l_@@_p_block_bool ,
6993   j .value_forbidden:n = true ,
6994   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6995   l .value_forbidden:n = true ,
6996   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6997   r .value_forbidden:n = true ,
6998   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6999   c .value_forbidden:n = true ,
7000   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7001   L .value_forbidden:n = true ,
7002   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7003   R .value_forbidden:n = true ,
7004   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7005   C .value_forbidden:n = true ,
7006   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7007   t .value_forbidden:n = true ,
7008   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7009   T .value_forbidden:n = true ,
7010   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7011   b .value_forbidden:n = true ,
7012   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7013   B .value_forbidden:n = true ,
7014   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7015   m .value_forbidden:n = true ,
7016   v-center .meta:n = m ,
7017   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7018   p .value_forbidden:n = true ,
7019   color .code:n =
7020     \@@_color:n { #1 }
7021     \tl_set_rescan:Nnn
7022       \l_@@_draw_tl
7023       { \char_set_catcode_other:N ! }
7024       { #1 } ,
7025   color .value_required:n = true ,
7026   respect-arraystretch .code:n =
7027     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7028   respect-arraystretch .value_forbidden:n = true ,
7029 }

```

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.

```

7030 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7031 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7032 {

```

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).

```

7033   \peek_remove_spaces:n

```

```

7034 {
7035   \tl_if_blank:nTF { #2 }
7036   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7037   {
7038     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7039     \@@_Block_i_czech \@@_Block_i
7040     #2 \q_stop
7041   }
7042   { #1 } { #3 } { #4 }
7043 }
7044 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7045 \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`.

```

7046 {
7047   \char_set_catcode_active:N -
7048   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7049 }

```

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.

```

7050 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7051 {

```

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).

```

7052   \bool_lazy_or:nnTF
7053   { \tl_if_blank_p:n { #1 } }
7054   { \str_if_eq_p:ee { * } { #1 } }
7055   { \int_set:Nn \l_tmpa_int { 100 } }
7056   { \int_set:Nn \l_tmpa_int { #1 } }
7057   \bool_lazy_or:nnTF
7058   { \tl_if_blank_p:n { #2 } }
7059   { \str_if_eq_p:ee { * } { #2 } }
7060   { \int_set:Nn \l_tmpb_int { 100 } }
7061   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7062   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7063   {
7064     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7065     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7066     { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7067   }
7068   { \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.

```

7069   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }

```

```

7070 \tl_set:N \l_tmpa_tl
7071 {
7072   { \int_use:N \c@iRow }
7073   { \int_use:N \c@jCol }
7074   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7075   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7076 }

```

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).

```

7077 \bool_set_false:N \l_tmpa_bool
7078 \bool_if:NT \l_@@_amp_in_blocks_bool

```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```

7079 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7080 \bool_case:nF
7081 {
7082   \l_tmpa_bool { \@@_Block_vii:eennn }
7083   \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.

```

7084 \l_@@_X_bool { \@@_Block_v:eennn }
7085 { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7086 { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7087 { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7088 }
7089 { \@@_Block_v:eennn }
7090 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7091 }

```

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.

```

7092 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7093 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7094 {
7095   \int_gincr:N \g_@@_block_box_int
7096   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7097   {
7098     \tl_gput_right:N \g_@@_pre_code_after_tl
7099     {
7100       \@@_actually_diagbox:nnnnnn
7101       { \int_use:N \c@iRow }
7102       { \int_use:N \c@jCol }
7103       { \int_eval:n { \c@iRow + #1 - 1 } }
7104       { \int_eval:n { \c@jCol + #2 - 1 } }
7105       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7106       { \g_@@_row_style_tl \exp_not:n { ##2 } }

```

```

7107     }
7108   }
7109   \box_gclear_new:c
7110   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7111   \hbox_gset:cn
7112   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7113   {

```

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}`).

```

7114   \tl_if_empty:NTF \l_@@_color_tl
7115   { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7116   { \@@_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`.

```

7117   \int_compare:nNnT { #1 } = \c_one_int
7118   {
7119     \int_if_zero:nTF \c@iRow
7120     {

```

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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7121   \cs_set_eq:NN \Block \@@_NullBlock:
7122   \l_@@_code_for_first_row_tl
7123   }
7124   {
7125     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7126     {
7127       \cs_set_eq:NN \Block \@@_NullBlock:
7128       \l_@@_code_for_last_row_tl
7129     }
7130   }
7131   \g_@@_row_style_tl
7132   }

```


The following command will be no-op when `respect-arraystretch` is in force.

```
7133 \@@_reset_arraystretch:
7134 \dim_zero:N \extrarowheight
```

`#4` is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```
7135 #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.).

```
7136 \@@_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}`.

```
7137 \bool_if:NTF \l_@@_tabular_bool
7138 {
7139   \bool_lazy_all:nTF
7140   {
7141     { \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.

```
7142   { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7143   { ! \g_@@_rotate_bool }
7144 }
```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```
7145 {
7146   \use:e
7147   {
```

The `\exp_not:N` is mandatory before `\begin`.

```
7148   \exp_not:N \begin { minipage }%
7149   [ \str_lowercase:o \l_@@_vpos_block_str ]
7150   { \l_@@_col_width_dim }
7151   \str_case:on \l_@@_hpos_block_str
7152   { c \centering r \raggedleft l \raggedright }
7153   }
7154   #5
7155   \end { minipage }
7156 }
```

In the other cases, we use a `{tabular}`.

```
7157 {
7158   \bool_if:NT \c_@@_testphase_table_bool
7159   { \tagpdfsetup { table / tagging = presentation } }
7160   \use:e
7161   {
7162     \exp_not:N \begin { tabular }%
7163     [ \str_lowercase:o \l_@@_vpos_block_str ]
7164     { @ { } \l_@@_hpos_block_str @ { } }
7165   }
7166   #5
7167   \end { tabular }
7168 }
7169 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
7170 {
7171   \c_math_toggle_token
7172   \use:e
7173   {
```

```

7174         \exp_not:N \begin { array }%
7175         [ \str_lowercase:o \l_@@_vpos_block_str ]
7176         { @ { } \l_@@_hpos_block_str @ { } }
7177     }
7178     #5
7179     \end { array }
7180     \c_math_toggle_token
7181 }
7182 }

```

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).

```

7183 \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.

```

7184 \int_compare:nNnT { #2 } = \c_one_int
7185 {
7186     \dim_gset:Nn \g_@@_blocks_wd_dim
7187     {
7188         \dim_max:nn
7189         \g_@@_blocks_wd_dim
7190         {
7191             \box_wd:c
7192             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7193         }
7194     }
7195 }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position.

```

7196 \bool_lazy_and:nnT
7197 { \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.

```

7198 { \str_if_empty_p:N \l_@@_vpos_block_str }
7199 {
7200     \dim_gset:Nn \g_@@_blocks_ht_dim
7201     {
7202         \dim_max:nn
7203         \g_@@_blocks_ht_dim
7204         {
7205             \box_ht:c
7206             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7207         }
7208     }
7209     \dim_gset:Nn \g_@@_blocks_dp_dim
7210     {
7211         \dim_max:nn
7212         \g_@@_blocks_dp_dim
7213         {
7214             \box_dp:c
7215             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7216         }
7217     }
7218 }
7219 \seq_gput_right:Ne \g_@@_blocks_seq
7220 {
7221     \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.

```

7222     {
7223         \exp_not:n { #3 } ,
7224         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7225         \bool_if:NT \g_@@_rotate_bool
7226         {
7227             \bool_if:NTF \g_@@_rotate_c_bool
7228             { m }
7229             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7230         }
7231     }
7232     {
7233         \box_use_drop:c
7234         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7235     }
7236 }
7237 \bool_set_false:N \g_@@_rotate_c_bool
7238 }

```

```

7239 \cs_new:Npn \@@_adjust_hpos_rotate:
7240 {
7241     \bool_if:NT \g_@@_rotate_bool
7242     {
7243         \str_set:Ne \l_@@_hpos_block_str
7244         {
7245             \bool_if:NTF \g_@@_rotate_c_bool
7246             { c }
7247             {
7248                 \str_case:onF \l_@@_vpos_block_str
7249                 { b l B l t r T r }
7250                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7251             }
7252         }
7253     }
7254 }

```

Despite its name the following command rotates the box of the block *but also does vertical ajustement of the baseline of the block*.

```

7255 \cs_new_protected:Npn \@@_rotate_box_of_block:
7256 {
7257     \box_grotate:cn
7258     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7259     { 90 }
7260     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7261     {
7262         \vbox_gset_top:cn
7263         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7264         {
7265             \skip_vertical:n { 0.8 ex }
7266             \box_use:c
7267             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7268         }
7269     }
7270     \bool_if:NT \g_@@_rotate_c_bool
7271     {
7272         \hbox_gset:cn
7273         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7274     {
7275         \c_math_toggle_token
7276         \vcenter
7277         {
7278             \box_use:c
7279             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7280         }
7281         \c_math_toggle_token
7282     }
7283 }
7284 }

```

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.

```

7285 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7286 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7287 {
7288     \seq_gput_right:Ne \g_@@_blocks_seq
7289     {
7290         \l_tmpa_tl
7291         { \exp_not:n { #3 } }
7292         {
7293             \bool_if:NTF \l_@@_tabular_bool
7294             {
7295                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7296     \@@_reset_arraystretch:
7297     \exp_not:n
7298     {
7299         \dim_zero:N \extrarowheight
7300         #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).

```

7301         \bool_if:NT \c_@@_testphase_table_bool
7302         { \tag_stop:n { table } }
7303         \use:e
7304         {
7305             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7306             { @ { } \l_@@_hpos_block_str @ { } }
7307         }
7308         #5
7309         \end { tabular }
7310     }
7311     \group_end:
7312 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7313     {
7314         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7315     \@@_reset_arraystretch:
7316     \exp_not:n
7317     {

```

```

7318         \dim_zero:N \extrarowheight
7319         #4
7320         \c_math_toggle_token
7321         \use:e
7322         {
7323             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7324             { @ { } \l_@@_hpos_block_str @ { } }
7325         }
7326         #5
7327         \end { array }
7328         \c_math_toggle_token
7329     }
7330 \group_end:
7331 }
7332 }
7333 }
7334 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7335 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7336 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7337 {
7338     \seq_gput_right:Ne \g_@@_blocks_seq
7339     {
7340         \l_tmpa_tl
7341         { \exp_not:n { #3 } }
7342         {
7343             \group_begin:
7344             \exp_not:n { #4 #5 }
7345             \group_end:
7346         }
7347     }
7348 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7349 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7350 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7351 {
7352     \seq_gput_right:Ne \g_@@_blocks_seq
7353     {
7354         \l_tmpa_tl
7355         { \exp_not:n { #3 } }
7356         { \exp_not:n { #4 #5 } }
7357     }
7358 }

```

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).

```

7359 \keys_define:nn { nicematrix / Block / SecondPass }
7360 {
7361     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7362     ampersand-in-blocks .default:n = true ,
7363     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7364     tikz .code:n =
7365         \IfPackageLoadedTF { tikz }
7366         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7367         { \@@_error:n { tikz~key~without~tikz } } ,
7368     tikz .value_required:n = true ,

```

```

7369 fill .code:n =
7370     \tl_set_rescan:Nnn
7371     \l_@@_fill_tl
7372     { \char_set_catcode_other:N ! }
7373     { #1 } ,
7374 fill .value_required:n = true ,
7375 opacity .tl_set:N = \l_@@_opacity_tl ,
7376 opacity .value_required:n = true ,
7377 draw .code:n =
7378     \tl_set_rescan:Nnn
7379     \l_@@_draw_tl
7380     { \char_set_catcode_other:N ! }
7381     { #1 } ,
7382 draw .default:n = default ,
7383 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7384 rounded-corners .default:n = 4 pt ,
7385 color .code:n =
7386     \@@_color:n { #1 }
7387     \tl_set_rescan:Nnn
7388     \l_@@_draw_tl
7389     { \char_set_catcode_other:N ! }
7390     { #1 } ,
7391 borders .clist_set:N = \l_@@_borders_clist ,
7392 borders .value_required:n = true ,
7393 hvlines .meta:n = { vlines , hlines } ,
7394 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7395 vlines .default:n = true ,
7396 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7397 hlines .default:n = true ,
7398 line-width .dim_set:N = \l_@@_line_width_dim ,
7399 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7400 j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7401     \bool_set_true:N \l_@@_p_block_bool ,
7402 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7403 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7404 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7405 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7406     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7407 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7408     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7409 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7410     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7411 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7412 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7413 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7414 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7415 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7416 m .value_forbidden:n = true ,
7417 v-center .meta:n = m ,
7418 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7419 p .value_forbidden:n = true ,
7420 name .tl_set:N = \l_@@_block_name_str ,
7421 name .value_required:n = true ,
7422 name .initial:n = ,
7423 respect-arraystretch .code:n =
7424     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7425 respect-arraystretch .value_forbidden:n = true ,
7426 transparent .bool_set:N = \l_@@_transparent_bool ,
7427 transparent .default:n = true ,
7428 transparent .initial:n = false ,
7429 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7430 }

```

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.

```

7431 \cs_new_protected:Npn \@@_draw_blocks:
7432 {
7433   \bool_if:NTF \c_@@_recent_array_bool
7434     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7435     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7436   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7437 }
7438 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
7439 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7440 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7441   \int_zero_new:N \l_@@_last_row_int
7442   \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

7443   \int_compare:nNnTF { #3 } > { 99 }
7444     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7445     { \int_set:Nn \l_@@_last_row_int { #3 } }
7446   \int_compare:nNnTF { #4 } > { 99 }
7447     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7448     { \int_set:Nn \l_@@_last_col_int { #4 } }
7449   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7450     {
7451       \bool_lazy_and:nnTF
7452         \l_@@_preamble_bool
7453         {
7454           \int_compare_p:n
7455             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7456         }
7457         {
7458           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7459           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7460           \@@_msg_redirect_name:nn { columns-not-used } { none }
7461         }
7462       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7463     }
7464   {
7465     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7466       { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7467       {
7468         \@@_Block_v:nneenn
7469           { #1 }
7470           { #2 }
7471           { \int_use:N \l_@@_last_row_int }
7472           { \int_use:N \l_@@_last_col_int }
7473           { #5 }
7474           { #6 }
7475       }
7476     }
7477   }

```

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

```
7478 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7479 {
```

The group is for the keys.

```
7480 \group_begin:
7481 \int_compare:nNt { #1 } = { #3 }
7482 { \str_set:Nn \l_@@_vpos_block_str { t } }
7483 \keys_set:nn { nicematrix / Block / SecondPass } { #5 }
```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster than `\str_if_in:nnT`.

```
7484 \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7485 \bool_lazy_and:nnT
7486 \l_@@_vlines_block_bool
7487 { ! \l_@@_ampersand_bool }
7488 {
7489   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7490   {
7491     \@@_vlines_block:nnn
7492     { \exp_not:n { #5 } }
7493     { #1 - #2 }
7494     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7495   }
7496 }
7497 \bool_if:NT \l_@@_hlines_block_bool
7498 {
7499   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7500   {
7501     \@@_hlines_block:nnn
7502     { \exp_not:n { #5 } }
7503     { #1 - #2 }
7504     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7505   }
7506 }
7507 \bool_if:NF \l_@@_transparent_bool
7508 {
7509   \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7510   {
```

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).

```
7511   \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7512   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7513 }
7514 }
```

```
7515 \tl_if_empty:NF \l_@@_draw_tl
7516 {
7517   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7518   { \@@_error:n { hlines-with-color } }
7519   \tl_gput_right:Ne \g_nicematrix_code_after_tl
7520   {
7521     \@@_stroke_block:nnn
```

#5 are the options

```
7522   { \exp_not:n { #5 } }
7523   { #1 - #2 }
7524   { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7525 }
7526 \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
```



```

7527         { { #1 } { #2 } { #3 } { #4 } }
7528     }
7529 \clist_if_empty:NF \l_@@_borders_clist
7530 {
7531     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7532     {
7533         \@@_stroke_borders_block:nnn
7534         { \exp_not:n { #5 } }
7535         { #1 - #2 }
7536         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7537     }
7538 }
7539 \tl_if_empty:NF \l_@@_fill_tl
7540 {
7541     \@@_add_opacity_to_fill:
7542     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7543     {
7544         \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
7545         { #1 - #2 }
7546         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7547         { \dim_use:N \l_@@_rounded_corners_dim }
7548     }
7549 }
7550 \seq_if_empty:NF \l_@@_tikz_seq
7551 {
7552     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7553     {
7554         \@@_block_tikz:nnnnn
7555         { \seq_use:Nn \l_@@_tikz_seq { , } }
7556         { #1 }
7557         { #2 }
7558         { \int_use:N \l_@@_last_row_int }
7559         { \int_use:N \l_@@_last_col_int }

```

We will have in that last field a list of lists of Tikz keys.

```

7560     }
7561 }
7562 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7563 {
7564     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7565     {
7566         \@@_actually_diagbox:nnnnn
7567         { #1 }
7568         { #2 }
7569         { \int_use:N \l_@@_last_row_int }
7570         { \int_use:N \l_@@_last_col_int }
7571         { \exp_not:n { ##1 } }
7572         { \exp_not:n { ##2 } }
7573     }
7574 }

```

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.

```
7575 \pgfpicture
7576 \pgfrememberpicturepositiononpagetrue
7577 \pgf@relevantforpicturesizefalse
7578 \@@_qpoint:n { row - #1 }
7579 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7580 \@@_qpoint:n { col - #2 }
7581 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7582 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7583 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7584 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7585 \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.

```
7586 \@@_pgf_rect_node:nnnnn
7587 { \@@_env: - #1 - #2 - block }
7588 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7589 \str_if_empty:NF \l_@@_block_name_str
7590 {
7591   \pgfnodealias
7592   { \@@_env: - \l_@@_block_name_str }
7593   { \@@_env: - #1 - #2 - block }
7594   \str_if_empty:NF \l_@@_name_str
7595   {
7596     \pgfnodealias
7597     { \l_@@_name_str - \l_@@_block_name_str }
7598     { \@@_env: - #1 - #2 - block }
7599   }
7600 }
```

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.

```
7601 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7602 {
7603   \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.

```
7604 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7605 {
```

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.

```
7606 \cs_if_exist:cT
7607 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7608 {
7609   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7610   {
7611     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
```

```

7612         \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7613     }
7614 }
7615 }

```

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.

```

7616     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7617     {
7618         \@@_qpoint:n { col - #2 }
7619         \dim_set_eq:NN \l_tmpb_dim \pgf@x
7620     }
7621     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7622     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7623     {
7624         \cs_if_exist:cT
7625         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7626         {
7627             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7628             {
7629                 \pgfpointanchor
7630                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7631                 { east }
7632                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7633             }
7634         }
7635     }
7636     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7637     {
7638         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7639         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7640     }
7641     \@@_pgf_rect_node:nnnnn
7642     { \@@_env: - #1 - #2 - block - short }
7643     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7644 }

```

If the creation of the “medium nodes” is required, we create a “medium node” for the block. The function `\@@_pgf_rect_node:nnn` takes in as arguments the name of the node and two PGF points.

```

7645     \bool_if:NT \l_@@_medium_nodes_bool
7646     {
7647         \@@_pgf_rect_node:nnn
7648         { \@@_env: - #1 - #2 - block - medium }
7649         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7650         {
7651             \pgfpointanchor
7652             { \@@_env:
7653               - \int_use:N \l_@@_last_row_int
7654               - \int_use:N \l_@@_last_col_int - medium
7655             }
7656             { south-east }
7657         }
7658     }
7659     \endpgfpicture

```

```

7660     \bool_if:NTF \l_@@_ampersand_bool
7661     {
7662         \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7663         \int_zero_new:N \l_@@_split_int
7664         \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7665         \pgfpicture
7666         \pgfrememberpicturepositiononpagetrue
7667         \pgf@relevantforpicturesizefalse

```

```

7668 \@@_qpoint:n { row - #1 }
7669 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7670 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7671 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7672 \@@_qpoint:n { col - #2 }
7673 \dim_set_eq:NN \l_tmpa_dim \pgf@x
7674 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7675 \dim_set:Nn \l_tmpb_dim
7676 { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7677 \bool_lazy_or:nnT
7678 \l_@@_vlines_block_bool
7679 { \str_if_eq_p:ee \l_@@_vlines_clist { all } }
7680 {
7681   \int_step_inline:nn { \l_@@_split_int - 1 }
7682   {
7683     \pgfpathmoveto
7684     {
7685       \pgfpoint
7686       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7687       \l_@@_tmpc_dim
7688     }
7689     \pgfpathlineto
7690     {
7691       \pgfpoint
7692       { \l_tmpa_dim + ##1 \l_tmpb_dim }
7693       \l_@@_tmpd_dim
7694     }
7695     \CT@arc@
7696     \pgfsetlinewidth { 1.1 \arrayrulewidth }
7697     \pgfsetrectcap
7698     \pgfusepathqstroke
7699   }
7700 }
7701 }
7702 \@@_qpoint:n { row - #1 - base }
7703 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7704 \int_step_inline:nn \l_@@_split_int
7705 {
7706   \group_begin:
7707   \dim_set:Nn \col@sep
7708   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7709   \pgftransformshift
7710   {
7711     \pgfpoint
7712     {
7713       \l_tmpa_dim + ##1 \l_tmpb_dim -
7714       \str_case:on \l_@@_hpos_block_str
7715       {
7716         l { \l_tmpb_dim + \col@sep }
7717         c { 0.5 \l_tmpb_dim }
7718         r { \col@sep }
7719       }
7720     }
7721     { \l_@@_tmpc_dim }
7722   }
7723   \pgfset { inner~sep = \c_zero_dim }
7724   \pgfnode
7725   { rectangle }
7726   {
7727     \str_case:on \l_@@_hpos_block_str
7728     {
7729       c { base }
7730       l { base~west }

```

```

7731         r { base~east }
7732     }
7733 }
7734 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7735 \group_end:
7736 }
7737 \endpgfpicture
7738 }

```

Now the case where there is no ampersand & in the content of the block.

```

7739 {
7740     \bool_if:NTF \l_@@_p_block_bool
7741     {

```

When the final user has used the key p, we have to compute the width.

```

7742     \pgfpicture
7743     \pgfrememberpicturepositiononpagetrue
7744     \pgf@relevantforpicturesizefalse
7745     \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7746     {
7747         \@@_qpoint:n { col - #2 }
7748         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7749         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7750     }
7751     {
7752         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7753         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7754         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7755     }
7756     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7757     \endpgfpicture
7758     \hbox_set:Nn \l_@@_cell_box
7759     {
7760         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7761         { \g_tmpb_dim }
7762         \str_case:on \l_@@_hpos_block_str
7763         { c \centering r \raggedleft l \raggedright j { } }
7764         #6
7765         \end { minipage }
7766     }
7767 }
7768 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7769 \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.

```

7770     \pgfpicture
7771     \pgfrememberpicturepositiononpagetrue
7772     \pgf@relevantforpicturesizefalse
7773     \bool_lazy_any:nTF
7774     {
7775         { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7776         { \str_if_eq_p:ee \l_@@_vpos_block_str { c } }
7777         { \str_if_eq_p:ee \l_@@_vpos_block_str { T } }
7778         { \str_if_eq_p:ee \l_@@_vpos_block_str { B } }
7779     }
7780     {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7781         \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 1.

```

7782         \bool_if:nT \g_@@_last_col_found_bool
7783         {
7784             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7785             { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_1_str }
7786         }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7787         \tl_set:Ne \l_tmpa_tl
7788         {
7789             \str_case:on \l_@@_vpos_block_str
7790             {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7791             { } { % added 2024-06-29
7792                 \str_case:on \l_@@_hpos_block_str
7793                 {
7794                     c { center }
7795                     l { west }
7796                     r { east }
7797                     j { center }
7798                 }
7799             }
7800         c {
7801             \str_case:on \l_@@_hpos_block_str
7802             {
7803                 c { center }
7804                 l { west }
7805                 r { east }
7806                 j { center }
7807             }
7808         }
7809         T {
7810             \str_case:on \l_@@_hpos_block_str
7811             {
7812                 c { north }
7813                 l { north-west }
7814                 r { north-east }
7815                 j { north }
7816             }
7817         }
7818         B {
7819             \str_case:on \l_@@_hpos_block_str
7820             {
7821                 c { south }
7822                 l { south-west }
7823                 r { south-east }
7824                 j { south }
7825             }
7826         }
7827     }
7828 }
7829 }
7830 }
7831 }
7832 \pgftransformshift
7833 {
7834     \pgfpointanchor
7835     {
7836         \@@_env: - #1 - #2 - block
7837         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7838     }
7839     { \l_tmpa_tl }

```

```

7840     }
7841     \pgfset { inner~sep = \c_zero_dim }
7842     \pgfnode
7843     { rectangle }
7844     { \l_tmpa_tl }
7845     { \box_use_drop:N \l_@@_cell_box } { } { }
7846 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

7847 {
7848     \pgfextracty \l_tmpa_dim
7849     {
7850         \@@_qpoint:n
7851         {
7852             row - \str_if_eq:eeTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7853             - base
7854         }
7855     }
7856     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7857     \pgfpointanchor
7858     {
7859         \@@_env: - #1 - #2 - block
7860         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7861     }
7862     {
7863         \str_case:on \l_@@_hpos_block_str
7864         {
7865             c { center }
7866             l { west }
7867             r { east }
7868             j { center }
7869         }
7870     }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7871     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7872     \pgfset { inner~sep = \c_zero_dim }
7873     \pgfnode
7874     { rectangle }
7875     {
7876         \str_case:on \l_@@_hpos_block_str
7877         {
7878             c { base }
7879             l { base~west }
7880             r { base~east }
7881             j { base }
7882         }
7883     }
7884     { \box_use_drop:N \l_@@_cell_box } { } { }
7885 }
7886 \endpgfpicture
7887 }
7888 \group_end:
7889 }

```

For the command `\cellcolor` used within a sub-cell of a `\Block` (when the character `&` is used inside the cell).

```

7890 \cs_set_protected:Npn \@@_fill:nnnnn #1 #2 #3 #4 #5
7891 {
7892     \pgfpicture
7893     \pgfrememberpicturepositiononpagetrue

```

```

7894 \pgf@relevantforpicturesizefalse
7895 \pgfpathrectanglecorners
7896 { \pgfpoint { #2 } { #3 } }
7897 { \pgfpoint { #4 } { #5 } }
7898 \pgfsetfillcolor { #1 }
7899 \pgfusepath { fill }
7900 \endpgfpicture
7901 }

```

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).

```

7902 \cs_new_protected:Npn \@@_add_opacity_to_fill:
7903 {
7904   \tl_if_empty:NF \l_@@_opacity_tl
7905   {
7906     \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7907       {
7908         \tl_set:Ne \l_@@_fill_tl
7909         {
7910           [ opacity = \l_@@_opacity_tl ,
7911           \tl_tail:o \l_@@_fill_tl
7912         }
7913       }
7914       {
7915         \tl_set:Ne \l_@@_fill_tl
7916         { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
7917       }
7918     }
7919   }

```

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).

```

7920 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7921 {
7922   \group_begin:
7923   \tl_clear:N \l_@@_draw_tl
7924   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7925   \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
7926   \pgfpicture
7927   \pgfrememberpicturepositiononpagetrue
7928   \pgf@relevantforpicturesizefalse
7929   \tl_if_empty:NF \l_@@_draw_tl
7930   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7931     \tl_if_eq:NnTF \l_@@_draw_tl { default }
7932     { \CT@arc@ }
7933     { \@@_color:o \l_@@_draw_tl }
7934   }
7935   \pgfsetcornersarced
7936   {
7937     \pgfpoint
7938     { \l_@@_rounded_corners_dim }
7939     { \l_@@_rounded_corners_dim }
7940   }
7941   \@@_cut_on_hyphen:w #2 \q_stop
7942   \int_compare:nNf \l_tmpa_tl > \c@iRow
7943   {
7944     \int_compare:nNf \l_tmpb_tl > \c@jCol

```



```

7945     {
7946         \@@_qpoint:n { row - \l_tmpa_tl }
7947         \dim_set_eq:NN \l_tmpb_dim \pgf@y
7948         \@@_qpoint:n { col - \l_tmpb_tl }
7949         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7950         \@@_cut_on_hyphen:w #3 \q_stop
7951         \int_compare:nNnT \l_tmpa_tl > \c@iRow
7952             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7953         \int_compare:nNnT \l_tmpb_tl > \c@jCol
7954             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7955         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7956         \dim_set_eq:NN \l_tmpa_dim \pgf@y
7957         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7958         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7959         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7960         \pgfpathrectanglecorners
7961             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7962             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7963         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7964             { \pgfusepathqstroke }
7965             { \pgfusepath { stroke } }
7966     }
7967 }
7968 \endpgfpicture
7969 \group_end:
7970 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7971 \keys_define:nn { nicematrix / BlockStroke }
7972 {
7973     color .tl_set:N = \l_@@_draw_tl ,
7974     draw .code:n =
7975         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7976     draw .default:n = default ,
7977     line-width .dim_set:N = \l_@@_line_width_dim ,
7978     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7979     rounded-corners .default:n = 4 pt
7980 }

```

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).

```

7981 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7982 {
7983     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7984     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
7985     \@@_cut_on_hyphen:w #2 \q_stop
7986     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7987     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7988     \@@_cut_on_hyphen:w #3 \q_stop
7989     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7990     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7991     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7992     {
7993         \use:e
7994         {
7995             \@@_vline:n
7996             {
7997                 position = ##1 ,
7998                 start = \l_@@_tmpc_tl ,
7999                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
8000                 total-width = \dim_use:N \l_@@_line_width_dim
8001             }

```

```

8002     }
8003   }
8004 }
8005 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8006 {
8007   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8008   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8009   \@@_cut_on_hyphen:w #2 \q_stop
8010   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8011   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8012   \@@_cut_on_hyphen:w #3 \q_stop
8013   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8014   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8015   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8016   {
8017     \use:e
8018     {
8019       \@@_hline:n
8020       {
8021         position = ##1 ,
8022         start = \l_@@_tmpd_tl ,
8023         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8024         total-width = \dim_use:N \l_@@_line_width_dim
8025       }
8026     }
8027   }
8028 }

```

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).

```

8029 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8030 {
8031   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8032   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8033   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8034   { \@@_error:n { borders~forbidden } }
8035   {
8036     \tl_clear_new:N \l_@@_borders_tikz_tl
8037     \keys_set:no
8038     { nicematrix / OnlyForTikzInBorders }
8039     \l_@@_borders_clist
8040     \@@_cut_on_hyphen:w #2 \q_stop
8041     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8042     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8043     \@@_cut_on_hyphen:w #3 \q_stop
8044     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8045     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8046     \@@_stroke_borders_block_i:
8047   }
8048 }
8049 \hook_gput_code:nnn { begindocument } { . }
8050 {
8051   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8052   {
8053     \c_@@_pgfortikzpicture_tl
8054     \@@_stroke_borders_block_ii:
8055     \c_@@_endpgfortikzpicture_tl
8056   }
8057 }
8058 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8059 {

```

```

8060 \pgfrememberpicturepositiononpagetrue
8061 \pgf@relevantforpicturesizefalse
8062 \CT@arc@
8063 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8064 \clist_if_in:NnT \l_@@_borders_clist { right }
8065 { \@@_stroke_vertical:n \l_tmpb_tl }
8066 \clist_if_in:NnT \l_@@_borders_clist { left }
8067 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8068 \clist_if_in:NnT \l_@@_borders_clist { bottom }
8069 { \@@_stroke_horizontal:n \l_tmpa_tl }
8070 \clist_if_in:NnT \l_@@_borders_clist { top }
8071 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8072 }
8073 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8074 {
8075   tikz .code:n =
8076     \cs_if_exist:NTF \tikzpicture
8077     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8078     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8079   tikz .value_required:n = true ,
8080   top .code:n = ,
8081   bottom .code:n = ,
8082   left .code:n = ,
8083   right .code:n = ,
8084   unknown .code:n = \@@_error:n { bad~border }
8085 }

```

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).

```

8086 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8087 {
8088   \@@_qpoint:n \l_@@_tmpc_tl
8089   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8090   \@@_qpoint:n \l_tmpa_tl
8091   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8092   \@@_qpoint:n { #1 }
8093   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8094   {
8095     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8096     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8097     \pgfusepath{stroke}
8098   }
8099   {
8100     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8101     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8102   }
8103 }

```

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).

```

8104 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8105 {
8106   \@@_qpoint:n \l_@@_tmpd_tl
8107   \clist_if_in:NnTF \l_@@_borders_clist { left }
8108   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8109   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8110   \@@_qpoint:n \l_tmpb_tl
8111   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8112   \@@_qpoint:n { #1 }
8113   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8114   {
8115     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8116     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }

```

```

8117     \pgfusepathqstroke
8118   }
8119   {
8120     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8121     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8122   }
8123 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8124 \keys_define:nn { nicematrix / BlockBorders }
8125 {
8126   borders .clist_set:N = \l_@@_borders_clist ,
8127   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8128   rounded-corners .default:n = 4 pt ,
8129   line-width .dim_set:N = \l_@@_line_width_dim
8130 }

```

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.

```

8131 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8132 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8133 {
8134   \begin { tikzpicture }
8135     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8136   \clist_map_inline:nn { #1 }
8137   {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8138     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8139     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8140     (
8141       [
8142         xshift = \dim_use:N \l_@@_offset_dim ,
8143         yshift = - \dim_use:N \l_@@_offset_dim
8144       ]
8145       #2 -| #3
8146     )
8147     rectangle
8148     (
8149       [
8150         xshift = - \dim_use:N \l_@@_offset_dim ,
8151         yshift = \dim_use:N \l_@@_offset_dim
8152       ]
8153       \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8154     ) ;
8155   }
8156   \end { tikzpicture }
8157 }

```

```

8158 \keys_define:nn { nicematrix / SpecialOffset }
8159 { 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.

```
8160 \cs_new_protected:Npn \@@_NullBlock:
8161 { \@@_collect_options:n { \@@_NullBlock_i: } }
8162 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8163 { }
```

27 How to draw the dotted lines transparently

```
8164 \cs_set_protected:Npn \@@_renew_matrix:
8165 {
8166   \RenewDocumentEnvironment { pmatrix } { } {
8167     { \pNiceMatrix }
8168     { \endpNiceMatrix }
8169   }
8170   \RenewDocumentEnvironment { vmatrix } { } {
8171     { \vNiceMatrix }
8172     { \endvNiceMatrix }
8173   }
8174   \RenewDocumentEnvironment { Vmatrix } { } {
8175     { \VNiceMatrix }
8176     { \endVNiceMatrix }
8177   }
8178   \RenewDocumentEnvironment { bmatrix } { } {
8179     { \bNiceMatrix }
8180     { \endbNiceMatrix }
8181   }
8182 }
```

28 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
8182 \keys_define:nn { nicematrix / Auto }
8183 {
8184   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8185   columns-type .value_required:n = true ,
8186   l .meta:n = { columns-type = l } ,
8187   r .meta:n = { columns-type = r } ,
8188   c .meta:n = { columns-type = c } ,
8189   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8190   delimiters / color .value_required:n = true ,
8191   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8192   delimiters / max-width .default:n = true ,
8193   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8194   delimiters .value_required:n = true ,
8195   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8196   rounded-corners .default:n = 4 pt
8197 }
8198 \NewDocumentCommand \AutoNiceMatrixWithDelims
8199 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8200 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8201 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8202 {
```

The group is for the protection of the keys.

```
8203   \group_begin:
8204   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
```

```

8205 \use:e
8206 {
8207   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8208   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8209   [ \exp_not:o \l_tmpa_tl ]
8210 }
8211 \int_if_zero:nT \l_@@_first_row_int
8212 {
8213   \int_if_zero:nT \l_@@_first_col_int { & }
8214   \prg_replicate:nn { #4 - 1 } { & }
8215   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8216 }
8217 \prg_replicate:nn { #3 }
8218 {
8219   \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).

```

8220   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8221   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8222 }
8223 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8224 {
8225   \int_if_zero:nT \l_@@_first_col_int { & }
8226   \prg_replicate:nn { #4 - 1 } { & }
8227   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8228 }
8229 \end { NiceArrayWithDelims }
8230 \group_end:
8231 }
8232 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8233 {
8234   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8235   {
8236     \bool_gset_true:N \g_@@_delims_bool
8237     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8238     \AutoNiceMatrixWithDelims { #2 } { #3 }
8239   }
8240 }
8241 \@@_define_com:nnn p ( )
8242 \@@_define_com:nnn b [ ]
8243 \@@_define_com:nnn v | |
8244 \@@_define_com:nnn V \ | \ |
8245 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8246 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8247 {
8248   \group_begin:
8249   \bool_gset_false:N \g_@@_delims_bool
8250   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8251   \group_end:
8252 }

```

29 The redefinition of the command \dotfill

```

8253 \cs_set_eq:NN \@@_old_dotfill \dotfill

```

```

8254 \cs_new_protected:Npn \@@_dotfill:
8255 {

```

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}`).

```

8256 \@@_old_dotfill
8257 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8258 }

```

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`.

```

8259 \cs_new_protected:Npn \@@_dotfill_i:
8260 { \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.

```

8261 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8262 {
8263   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8264   {
8265     \@@_actually_diagbox:nnnnnn
8266     { \int_use:N \c@iRow }
8267     { \int_use:N \c@jCol }
8268     { \int_use:N \c@iRow }
8269     { \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.

```

8270   { \g_@@_row_style_tl \exp_not:n { #1 } }
8271   { \g_@@_row_style_tl \exp_not:n { #2 } }
8272 }

```

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`.

```

8273   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8274   {
8275     { \int_use:N \c@iRow }
8276     { \int_use:N \c@jCol }
8277     { \int_use:N \c@iRow }
8278     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8279     { }
8280   }
8281 }

```

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.

```

8282 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8283 {
8284   \pgfpicture
8285   \pgf@relevantforpicturesizefalse
8286   \pgfrememberpicturepositiononpagetrue
8287   \@@_qpoint:n { row - #1 }

```

```

8288 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8289 \@@_qpoint:n { col - #2 }
8290 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8291 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8292 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8293 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8294 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8295 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8296 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8297 {

```

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.

```

8298 \CT@arc@
8299 \pgfsetroundcap
8300 \pgfusepathqstroke
8301 }
8302 \pgfset { inner~sep = 1 pt }
8303 \pgfscope
8304 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8305 \pgfnode { rectangle } { south~west }
8306 {
8307 \begin { minipage } { 20 cm }

```

The `\scan_stop:` avoids an error in math mode when the argument #5 is empty.

```

8308 \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
8309 \end { minipage }
8310 }
8311 { }
8312 { }
8313 \endpgfscope
8314 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8315 \pgfnode { rectangle } { north~east }
8316 {
8317 \begin { minipage } { 20 cm }
8318 \raggedleft
8319 \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
8320 \end { minipage }
8321 }
8322 { }
8323 { }
8324 \endpgfpicture
8325 }

```

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`.

```

8326 \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 `\`.

```

8327 \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`.

```

8328 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8329 {
8330   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8331   \@@_CodeAfter_iv:n
8332 }

```

We catch the argument of the command `\end` (in `#1`).

```

8333 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8334 {

```

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.

```

8335   \str_if_eq:eeTF \currenvir { #1 }
8336   { \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`.

```

8337   {
8338     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8339     \@@_CodeAfter_ii:n
8340   }
8341 }

```

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 columnn. 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.

```

8342 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8343 {
8344   \pgfpicture
8345   \pgfrememberpicturepositiononpagetrue
8346   \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.

```

8347   \@@_qpoint:n { row - 1 }
8348   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8349   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8350   \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).

```

8351   \bool_if:nTF { #3 }
8352   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8353   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8354   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8355   {
8356     \cs_if_exist:cT
8357     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }

```

```

8358     {
8359         \pgfpointanchor
8360         { \l_@@_env: - #1 - #2 }
8361         { \bool_if:nTF { #3 } { west } { east } }
8362         \dim_set:Nn \l_tmpa_dim
8363         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf{x }
8364     }
8365 }

```

Now we can put the delimiter with a node of PGF.

```

8366     \pgfset { inner~sep = \c_zero_dim }
8367     \dim_zero:N \nulldelimiterspace
8368     \pgftransformshift
8369     {
8370         \pgfpoint
8371         { \l_tmpa_dim }
8372         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8373     }
8374     \pgfnode
8375     { rectangle }
8376     { \bool_if:nTF { #3 } { east } { west } }
8377     {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8378         \nullfont
8379         \c_math_toggle_token
8380         \l_@@_color:o \l_@@_delimiters_color_tl
8381         \bool_if:nTF { #3 } { \left #1 } { \left . }
8382         \vcenter
8383         {
8384             \nullfont
8385             \hrule \@height
8386                 \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8387                 \@depth \c_zero_dim
8388                 \@width \c_zero_dim
8389         }
8390         \bool_if:nTF { #3 } { \right . } { \right #1 }
8391         \c_math_toggle_token
8392     }
8393     { }
8394     { }
8395     \endpgfpicture
8396 }

```

33 The command `\SubMatrix`

```

8397 \keys_define:nn { nicematrix / sub-matrix }
8398 {
8399     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8400     extra-height .value_required:n = true ,
8401     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8402     left-xshift .value_required:n = true ,
8403     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8404     right-xshift .value_required:n = true ,
8405     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8406     xshift .value_required:n = true ,
8407     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8408     delimiters / color .value_required:n = true ,
8409     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8410     slim .default:n = true ,
8411     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,

```

```

8412     hlines .default:n = all ,
8413     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8414     vlines .default:n = all ,
8415     hvlines .meta:n = { hlines, vlines } ,
8416     hvlines .value_forbidden:n = true
8417   }
8418   \keys_define:nn { nicematrix }
8419   {
8420     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8421     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8422     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8423     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8424   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8425   \keys_define:nn { nicematrix / SubMatrix }
8426   {
8427     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8428     delimiters / color .value_required:n = true ,
8429     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8430     hlines .default:n = all ,
8431     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8432     vlines .default:n = all ,
8433     hvlines .meta:n = { hlines, vlines } ,
8434     hvlines .value_forbidden:n = true ,
8435     name .code:n =
8436       \tl_if_empty:nTF { #1 }
8437       { \@@_error:n { Invalid-name } }
8438       {
8439         \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8440         {
8441           \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8442           { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8443           {
8444             \str_set:Nn \l_@@_submatrix_name_str { #1 }
8445             \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8446           }
8447         }
8448         { \@@_error:n { Invalid-name } }
8449       } ,
8450     name .value_required:n = true ,
8451     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8452     rules .value_required:n = true ,
8453     code .tl_set:N = \l_@@_code_tl ,
8454     code .value_required:n = true ,
8455     unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8456   }

8457   \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8458   {
8459     \peek_remove_spaces:n
8460     {
8461       \tl_gput_right:Ne \g_@@_pre_code_after_tl
8462       {
8463         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8464         [
8465           delimiters / color = \l_@@_delimiters_color_tl ,
8466           hlines = \l_@@_submatrix_hlines_clist ,
8467           vlines = \l_@@_submatrix_vlines_clist ,
8468           extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8469           left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8470           right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,

```

```

8471         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8472         #5
8473     ]
8474 }
8475 \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8476 }
8477 }
8478 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8479 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8480 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8481 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8482 {
8483     \seq_gput_right:Ne \g_@@_submatrix_seq
8484     {

```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8485     { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8486     { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8487     { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8488     { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8489 }
8490 }

```

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`.

```

8491 \hook_gput_code:nnn { begindocument } { . }
8492 {
8493     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8494     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8495     \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8496     {
8497         \peek_remove_spaces:n
8498         {
8499             \@@_sub_matrix:nnnnnnn
8500             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8501         }
8502     }
8503 }

```

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).

```

8504 \NewDocumentCommand \@@_compute_i_j:nn
8505 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8506 { \@@_compute_i_j:nnnn #1 #2 }

```

```

8507 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8508 {
8509   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8510   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8511   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8512   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8513   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8514     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8515   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8516     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8517   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8518     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8519   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8520     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8521 }
8522 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8523 {
8524   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8525   \@@_compute_i_j:nn { #2 } { #3 }
8526   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8527     { \cs_set_nopar:Npn \arraystretch { 1 } }
8528   \bool_lazy_or:nnTF
8529     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8530     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8531     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8532   {
8533     \str_clear_new:N \l_@@_submatrix_name_str
8534     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8535     \pgfpicture
8536     \pgfrememberpicturepositiononpagetrue
8537     \pgf@relevantforpicturesizefalse
8538     \pgfset { inner~sep = \c_zero_dim }
8539     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8540     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

8541   \bool_if:NTF \l_@@_submatrix_slim_bool
8542     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8543     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8544     {
8545       \cs_if_exist:cT
8546         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8547       {
8548         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8549         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8550           { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8551       }
8552       \cs_if_exist:cT
8553         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8554       {
8555         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8556         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8557           { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8558       }
8559     }
8560   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8561     { \@@_error:nn { Impossible~delimiter } { left } }
8562   {
8563     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8564       { \@@_error:nn { Impossible~delimiter } { right } }
8565     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8566   }

```

```

8567         \endpgfpicture
8568     }
8569 \group_end:
8570 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8571 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8572 {
8573     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8574     \dim_set:Nn \l_@@_y_initial_dim
8575     {
8576         \fp_to_dim:n
8577         {
8578             \pgf@y
8579             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8580         }
8581     }
8582     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8583     \dim_set:Nn \l_@@_y_final_dim
8584     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8585     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8586     {
8587         \cs_if_exist:cT
8588         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8589         {
8590             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8591             \dim_set:Nn \l_@@_y_initial_dim
8592             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8593         }
8594         \cs_if_exist:cT
8595         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8596         {
8597             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8598             \dim_compare:nNt \pgf@y < \l_@@_y_final_dim
8599             { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8600         }
8601     }
8602     \dim_set:Nn \l_tmpa_dim
8603     {
8604         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8605         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8606     }
8607     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8608     \group_begin:
8609     \pgfsetlinewidth { 1.1 \arrayrulewidth }
8610     \@@_set_CT@arc@:o \l_@@_rules_color_tl
8611     \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`.

```

8612     \seq_map_inline:Nn \g_@@_cols_vlism_seq
8613     {
8614         \int_compare:nNt \l_@@_first_j_tl < { ##1 }
8615         {
8616             \int_compare:nNt
8617             { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8618             {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8619         \@@_qpoint:n { col - ##1 }

```

```

8620         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8621         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8622         \pgfusepathqstroke
8623     }
8624 }
8625 }

```

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.

```

8626 \str_if_eq:eeTF \l_@@_submatrix_vlines_clist { all }
8627 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8628 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8629 {
8630     \bool_lazy_and:nnTF
8631     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8632     {
8633         \int_compare_p:nNn
8634         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8635     {
8636         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8637         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8638         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8639         \pgfusepathqstroke
8640     }
8641     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8642 }

```

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.

```

8643 \str_if_eq:eeTF \l_@@_submatrix_hlines_clist { all }
8644 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8645 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8646 {
8647     \bool_lazy_and:nnTF
8648     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8649     {
8650         \int_compare_p:nNn
8651         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8652     {
8653         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8654     \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8655     \dim_set:Nn \l_tmpa_dim
8656     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8657     \str_case:nn { #1 }
8658     {
8659         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8660         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8661         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8662     }
8663     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8664     \dim_set:Nn \l_tmpb_dim
8665     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8666     \str_case:nn { #2 }
8667     {
8668         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8669         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8670         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8671     }

```

```

8672         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8673         \pgfusepathqstroke
8674     \group_end:
8675 }
8676 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
8677 }

```

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).

```

8678 \str_if_empty:NF \l_@@_submatrix_name_str
8679 {
8680     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8681     \l_@@_x_initial_dim \l_@@_y_initial_dim
8682     \l_@@_x_final_dim \l_@@_y_final_dim
8683 }
8684 \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`.

```

8685 \begin { pgfscope }
8686 \pgftransformshift
8687 {
8688     \pgfpoint
8689     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8690     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8691 }
8692 \str_if_empty:NTF \l_@@_submatrix_name_str
8693 { \@@_node_left:nn #1 { } }
8694 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8695 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8696 \pgftransformshift
8697 {
8698     \pgfpoint
8699     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8700     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8701 }
8702 \str_if_empty:NTF \l_@@_submatrix_name_str
8703 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8704 {
8705     \@@_node_right:nnnn #2
8706     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8707 }
8708 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8709 \flag_clear_new:N \l_@@_code_flag
8710 \l_@@_code_tl
8711 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-lj$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8712 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.


```

8713 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8714 {
8715   \use:e
8716   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8717 }

```

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 -.

```

8718 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8719 { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

8720 \tl_const:Nn \c_@@_integers_alist_tl
8721 {
8722   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8723   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8724   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8725   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8726 }

8727 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8728 {

```

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`.

```

8729   \tl_if_empty:nTF { #2 }
8730   {
8731     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8732     {
8733       \flag_raise:N \l_@@_code_flag
8734       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8735       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8736       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8737     }
8738     { #1 }
8739   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8740     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8741   }

```

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`).

```

8742 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8743 {
8744   \str_case:nnF { #1 }
8745   {
8746     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8747     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8748   }

```

Now the case of a node of the form $i-j$.

```

8749   {
8750     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8751     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8752   }
8753 }

```

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`).

```

8754 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8755 {
8756   \pgfnode
8757   { rectangle }
8758   { east }
8759   {
8760     \nullfont
8761     \c_math_toggle_token
8762     \@@_color:o \l_@@_delimiters_color_tl
8763     \left #1
8764     \vcenter
8765     {
8766       \nullfont
8767       \hrule \@height \l_tmpa_dim
8768               \@depth \c_zero_dim
8769               \@width \c_zero_dim
8770     }
8771     \right .
8772     \c_math_toggle_token
8773   }
8774   { #2 }
8775   { }
8776 }

```

The command `\@@_node_right:nnn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

8777 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8778 {
8779   \pgfnode
8780   { rectangle }
8781   { west }
8782   {
8783     \nullfont
8784     \c_math_toggle_token
8785     \colorlet { current-color } { . }
8786     \@@_color:o \l_@@_delimiters_color_tl
8787     \left .
8788     \vcenter
8789     {
8790       \nullfont
8791       \hrule \@height \l_tmpa_dim
8792               \@depth \c_zero_dim
8793               \@width \c_zero_dim
8794     }
8795     \right #1
8796     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8797     ^ { \color { current-color } \smash { #4 } }
8798     \c_math_toggle_token
8799   }
8800   { #2 }
8801   { }
8802 }

```

34 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8803 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8804 {
8805   \peek_remove_spaces:n
8806   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8807 }

8808 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8809 {
8810   \peek_remove_spaces:n
8811   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8812 }

8813 \keys_define:nn { nicematrix / Brace }
8814 {
8815   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8816   left-shorten .default:n = true ,
8817   left-shorten .value_forbidden:n = true ,
8818   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8819   right-shorten .default:n = true ,
8820   right-shorten .value_forbidden:n = true ,
8821   shorten .meta:n = { left-shorten , right-shorten } ,
8822   shorten .value_forbidden:n = true ,
8823   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8824   yshift .value_required:n = true ,
8825   yshift .initial:n = \c_zero_dim ,
8826   color .tl_set:N = \l_tmpa_tl ,
8827   color .value_required:n = true ,
8828   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8829 }

```

#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`.

```

8830 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8831 {
8832   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8833 \@@_compute_i_j:nn { #1 } { #2 }
8834 \bool_lazy_or:nnTF
8835   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8836   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8837   {
8838     \str_if_eq:eeTF { #5 } { under }
8839     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8840     { \@@_error:nn { Construct-too-large } { \OverBrace } }
8841   }
8842   {
8843     \tl_clear:N \l_tmpa_tl
8844     \keys_set:nn { nicematrix / Brace } { #4 }
8845     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8846     \pgfpicture
8847     \pgfrememberpicturepositiononpagetrue
8848     \pgf@relevantforpicturesizefalse
8849     \bool_if:NT \l_@@_brace_left_shorten_bool
8850     {
8851       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8852       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8853       {

```

```

8854         \cs_if_exist:cT
8855         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_first_j_tl }
8856         {
8857             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_first_j_tl } { west }
8858
8859             \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8860             { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8861         }
8862     }
8863 }
8864 \bool_lazy_or:nnT
8865 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8866 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8867 {
8868     \l_@@_qpoint:n { col - \l_@@_first_j_tl }
8869     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8870 }
8871 \bool_if:NT \l_@@_brace_right_shorten_bool
8872 {
8873     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8874     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8875     {
8876         \cs_if_exist:cT
8877         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
8878         {
8879             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
8880             \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8881             { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8882         }
8883     }
8884 }
8885 \bool_lazy_or:nnT
8886 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8887 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8888 {
8889     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8890     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8891 }
8892 \pgfset { inner~sep = \c_zero_dim }
8893 \str_if_eq:eeTF { #5 } { under }
8894 { \l_@@_underbrace_i:n { #3 } }
8895 { \l_@@_overbrace_i:n { #3 } }
8896 \endpgfpicture
8897 }
8898 \group_end:
8899 }

```

The argument is the text to put above the brace.

```

8900 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
8901 {
8902     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
8903     \pgftransformshift
8904     {
8905         \pgfpoint
8906         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8907         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8908     }
8909     \pgfnode
8910     { rectangle }
8911     { south }
8912     {
8913         \vtop
8914         {
8915             \group_begin:

```

```

8916     \everycr { }
8917     \halign
8918     {
8919         \hfil ## \hfil \crcr
8920         \bool_if:NTF \l_@@_tabular_bool
8921         { \begin { tabular } { c } #1 \end { tabular } }
8922         { $ \begin { array } { c } #1 \end { array } $ }
8923     \cr
8924     \c_math_toggle_token
8925     \overbrace
8926     {
8927         \hbox_to_wd:nn
8928         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8929         { }
8930     }
8931     \c_math_toggle_token
8932     \cr
8933     }
8934     \group_end:
8935 }
8936 }
8937 { }
8938 { }
8939 }

```

The argument is the text to put under the brace.

```

8940 \cs_new_protected:Npn \@@_underbrace_i:n #1
8941 {
8942     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8943     \pgftransformshift
8944     {
8945         \pgfpoint
8946         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8947         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8948     }
8949     \pgfnode
8950     { rectangle }
8951     { north }
8952     {
8953         \group_begin:
8954         \everycr { }
8955         \vbox
8956         {
8957             \halign
8958             {
8959                 \hfil ## \hfil \crcr
8960                 \c_math_toggle_token
8961                 \underbrace
8962                 {
8963                     \hbox_to_wd:nn
8964                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8965                     { }
8966                 }
8967                 \c_math_toggle_token
8968                 \cr
8969                 \bool_if:NTF \l_@@_tabular_bool
8970                 { \begin { tabular } { c } #1 \end { tabular } }
8971                 { $ \begin { array } { c } #1 \end { array } $ }
8972                 \cr
8973             }
8974         }
8975         \group_end:
8976     }
8977     { }

```

```

8978     { }
8979 }

```

35 The command TikzEveryCell

```

8980 \bool_new:N \l_@@_not_empty_bool
8981 \bool_new:N \l_@@_empty_bool
8982
8983 \keys_define:nn { nicematrix / TikzEveryCell }
8984 {
8985     not-empty .code:n =
8986         \bool_lazy_or:nnTF
8987             \l_@@_in_code_after_bool
8988             \g_@@_recreate_cell_nodes_bool
8989             { \bool_set_true:N \l_@@_not_empty_bool }
8990             { \@@_error:n { detection~of~empty~cells } } ,
8991     not-empty .value_forbidden:n = true ,
8992     empty .code:n =
8993         \bool_lazy_or:nnTF
8994             \l_@@_in_code_after_bool
8995             \g_@@_recreate_cell_nodes_bool
8996             { \bool_set_true:N \l_@@_empty_bool }
8997             { \@@_error:n { detection~of~empty~cells } } ,
8998     empty .value_forbidden:n = true ,
8999     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9000 }
9001
9002
9003 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9004 {
9005     \IfPackageLoadedTF { tikz }
9006     {
9007         \group_begin:
9008         \keys_set:nn { nicematrix / TikzEveryCell } { #1 }

```

The inner pair of braces in the following line is mandatory because, the last argument of `\@@_tikz:nnnnn` is a *list of lists* of TikZ keys.

```

9009         \tl_set:Nn \l_tmpa_tl { { #2 } }
9010         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9011             { \@@_for_a_block:nnnnn ##1 }
9012         \@@_all_the_cells:
9013         \group_end:
9014     }
9015     { \@@_error:n { TikzEveryCell~without~tikz } }
9016 }
9017
9018 \tl_new:N \@@_i_tl
9019 \tl_new:N \@@_j_tl
9020
9021
9022 \cs_new_protected:Nn \@@_all_the_cells:
9023 {
9024     \int_step_variable:nNn \c@iRow \@@_i_tl
9025     {
9026         \int_step_variable:nNn \c@jCol \@@_j_tl
9027         {
9028             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9029             {
9030                 \clist_if_in:NcF \l_@@_corners_cells_clist
9031                     { \@@_i_tl - \@@_j_tl }

```

```

9032         {
9033             \bool_set_false:N \l_tmpa_bool
9034             \cs_if_exist:cTF
9035             { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9036             {
9037                 \bool_if:NF \l_@@_empty_bool
9038                 { \bool_set_true:N \l_tmpa_bool }
9039             }
9040             {
9041                 \bool_if:NF \l_@@_not_empty_bool
9042                 { \bool_set_true:N \l_tmpa_bool }
9043             }
9044             \bool_if:NT \l_tmpa_bool
9045             {
9046                 \@@_block_tikz:onnnn
9047                 \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9048             }
9049         }
9050     }
9051 }
9052 }
9053 }
9054
9055 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9056 {
9057     \bool_if:NF \l_@@_empty_bool
9058     {
9059         \@@_block_tikz:onnnn
9060         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9061     }
9062     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9063 }
9064
9065 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9066 {
9067     \int_step_inline:nnn { #1 } { #3 }
9068     {
9069         \int_step_inline:nnn { #2 } { #4 }
9070         { \cs_set_nopar:cpn { cell - ##1 - ###1 } { } }
9071     }
9072 }

```

36 The command \ShowCellNames

```

9073 \NewDocumentCommand \@@_ShowCellNames { }
9074 {
9075     \bool_if:NT \l_@@_in_code_after_bool
9076     {
9077         \pgfpicture
9078         \pgfrememberpicturepositiononpagetrue
9079         \pgf@relevantforpicturesizefalse
9080         \pgfpathrectanglecorners
9081         { \@@_qpoint:n { 1 } }
9082         {
9083             \@@_qpoint:n
9084             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9085         }
9086         \pgfsetfillopacity { 0.75 }
9087         \pgfsetfillcolor { white }
9088         \pgfusepathqfill
9089         \endpgfpicture
9090     }

```

```

9091 \dim_gzero_new:N \g_@@_tmpc_dim
9092 \dim_gzero_new:N \g_@@_tmpd_dim
9093 \dim_gzero_new:N \g_@@_tmpe_dim
9094 \int_step_inline:nn \c@iRow
9095 {
9096   \bool_if:NTF \l_@@_in_code_after_bool
9097   {
9098     \pgfpicture
9099     \pgfrememberpicturepositiononpagetrue
9100     \pgf@relevantforpicturesizefalse
9101   }
9102   { \begin { pgfpicture } }
9103   \@@_qpoint:n { row - ##1 }
9104   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9105   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9106   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9107   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9108   \bool_if:NTF \l_@@_in_code_after_bool
9109   { \endpgfpicture }
9110   { \end { pgfpicture } }
9111   \int_step_inline:nn \c@jCol
9112   {
9113     \hbox_set:Nn \l_tmpa_box
9114     {
9115       \normalfont \Large \sffamily \bfseries
9116       \bool_if:NTF \l_@@_in_code_after_bool
9117       { \color { red } }
9118       { \color { red ! 50 } }
9119       ##1 - ####1
9120     }
9121     \bool_if:NTF \l_@@_in_code_after_bool
9122     {
9123       \pgfpicture
9124       \pgfrememberpicturepositiononpagetrue
9125       \pgf@relevantforpicturesizefalse
9126     }
9127     { \begin { pgfpicture } }
9128     \@@_qpoint:n { col - ####1 }
9129     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9130     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9131     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9132     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9133     \bool_if:NTF \l_@@_in_code_after_bool
9134     { \endpgfpicture }
9135     { \end { pgfpicture } }
9136     \fp_set:Nn \l_tmpa_fp
9137     {
9138       \fp_min:nn
9139       {
9140         \fp_min:nn
9141         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9142         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9143       }
9144       { 1.0 }
9145     }
9146     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9147     \pgfpicture
9148     \pgfrememberpicturepositiononpagetrue
9149     \pgf@relevantforpicturesizefalse
9150     \pgftransformshift
9151     {
9152       \pgfpoint
9153       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }

```



```

9154         { \dim_use:N \g_tmpa_dim }
9155     }
9156     \pgfnode
9157     { rectangle }
9158     { center }
9159     { \box_use:N \l_tmpa_box }
9160     { }
9161     { }
9162     \endpgfpicture
9163 }
9164 }
9165 }

```

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.

```

9166 \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.

```

9167 \bool_new:N \g_@@_footnote_bool

9168 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9169 {
9170   The~key~'\l_keys_key_str'~is~unknown. \\
9171   That~key~will~be~ignored. \\
9172   For~a~list~of~the~available~keys,~type~H~<return>.
9173 }
9174 {
9175   The~available~keys~are~(in~alphabetic~order):~
9176   footnote,~
9177   footnotehyper,~
9178   messages-for-Overleaf,~
9179   renew-dots,~and~
9180   renew-matrix.
9181 }

9182 \keys_define:nn { nicematrix / Package }
9183 {
9184   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9185   renew-dots .value_forbidden:n = true ,
9186   renew-matrix .code:n = \@@_renew_matrix: ,
9187   renew-matrix .value_forbidden:n = true ,
9188   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9189   footnote .bool_set:N = \g_@@_footnote_bool ,
9190   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,

```

The test for a potential modification of array has been deleted. You keep the following key only for compatibility but maybe we will delete it.

```

9191   no-test-for-array .code:n = \prg_do_nothing: ,
9192   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9193 }
9194 \ProcessKeysOptions { nicematrix / Package }

```

```

9195 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9196 {
9197   You~can't~use~the~option~'footnote'~because~the~package~
9198   footnotehyper~has~already~been~loaded.~
9199   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9200   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9201   of~the~package~footnotehyper.\\
9202   The~package~footnote~won't~be~loaded.
9203 }
9204 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9205 {
9206   You~can't~use~the~option~'footnotehyper'~because~the~package~
9207   footnote~has~already~been~loaded.~
9208   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9209   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9210   of~the~package~footnote.\\
9211   The~package~footnotehyper~won't~be~loaded.
9212 }
9213 \bool_if:NT \g_@@_footnote_bool
9214 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9215   \IfClassLoadedTF { beamer }
9216   { \bool_set_false:N \g_@@_footnote_bool }
9217   {
9218     \IfPackageLoadedTF { footnotehyper }
9219     { \@@_error:n { footnote-with-footnotehyper-package } }
9220     { \usepackage { footnote } }
9221   }
9222 }
9223 \bool_if:NT \g_@@_footnotehyper_bool
9224 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9225   \IfClassLoadedTF { beamer }
9226   { \bool_set_false:N \g_@@_footnote_bool }
9227   {
9228     \IfPackageLoadedTF { footnote }
9229     { \@@_error:n { footnotehyper-with-footnote-package } }
9230     { \usepackage { footnotehyper } }
9231   }
9232   \bool_set_true:N \g_@@_footnote_bool
9233 }

```

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.

```

9234 \bool_new:N \l_@@_underscore_loaded_bool
9235 \IfPackageLoadedT { underscore }
9236 { \bool_set_true:N \l_@@_underscore_loaded_bool }

```

```

9237 \hook_gput_code:nnn { begindocument } { . }
9238 {
9239   \bool_if:NF \l_@@_underscore_loaded_bool
9240   {
9241     \IfPackageLoadedT { underscore }
9242     { \@@_error:n { underscore~after~nicematrix } }
9243   }
9244 }

```

39 Error messages of the package

```

9245 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9246 { \str_const:Nn \c_@@_available_keys_str { } }
9247 {
9248   \str_const:Nn \c_@@_available_keys_str
9249   { For~a~list~of~the~available~keys,~type~H~<return>. }
9250 }
9251 \seq_new:N \g_@@_types_of_matrix_seq
9252 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9253 {
9254   NiceMatrix ,
9255   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9256 }
9257 \seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9258 { \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`.

```

9259 \cs_new_protected:Npn \@@_error_too_much_cols:
9260 {
9261   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9262   { \@@_fatal:nn { too~much~cols~for~array } }
9263   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9264   { \@@_fatal:n { too~much~cols~for~matrix } }
9265   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9266   { \@@_fatal:n { too~much~cols~for~matrix } }
9267   \bool_if:NF \l_@@_last_col_without_value_bool
9268   { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9269 }

```

The following command must *not* be protected since it's used in an error message.

```

9270 \cs_new:Npn \@@_message_hdotsfor:
9271 {
9272   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9273   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9274 }
9275 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9276 {
9277   Incompatible~options.\\
9278   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9279   The~output~will~not~be~reliable.
9280 }
9281 \@@_msg_new:nn { negative-weight }
9282 {
9283   Negative~weight.\\
9284   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~

```

```

9285     the-value~'\int_use:N \l_@@_weight_int'.\\
9286     The-absolute-value-will-be-used.
9287 }
9288 \@@_msg_new:nn { last-col-not-used }
9289 {
9290     Column-not-used.\\
9291     The-key~'last-col'~is-in-force-but-you-have-not-used-that-last-column~
9292     in-your~\@@_full_name_env:.~However,~you~can~go~on.
9293 }
9294 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9295 {
9296     Too-much-columns.\\
9297     In-the-row~\int_eval:n { \c@iRow },~
9298     you-try-to-use-more-columns~
9299     than-allowed-by-your~\@@_full_name_env:.~\@@_message_hdotsfor:\
9300     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9301     (plus-the-exterior-columns).~This-error-is-fatal.
9302 }
9303 \@@_msg_new:nn { too-much-cols-for-matrix }
9304 {
9305     Too-much-columns.\\
9306     In-the-row~\int_eval:n { \c@iRow },~
9307     you-try-to-use-more-columns-than-allowed-by-your~
9308     \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall~that~the-maximal~
9309     number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
9310     columns)~is-fixed-by-the-LaTeX-counter~'MaxMatrixCols'.~
9311     Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9312     \token_to_str:N \setcounter\ to-change-that-value).~
9313     This-error-is-fatal.
9314 }
9315 \@@_msg_new:nn { too-much-cols-for-array }
9316 {
9317     Too-much-columns.\\
9318     In-the-row~\int_eval:n { \c@iRow },~
9319     ~you-try-to-use-more-columns-than-allowed-by-your~
9320     \@@_full_name_env:.~\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9321     \int_use:N \g_@@_static_num_of_col_int\
9322     ~(plus-the-potential-exterior-ones).~
9323     This-error-is-fatal.
9324 }
9325 \@@_msg_new:nn { columns-not-used }
9326 {
9327     Columns-not-used.\\
9328     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9329     \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9330     The-columns-you-did-not-used-won't-be-created.\\
9331     You-won't-have-similar-error-message-till-the-end-of-the-document.
9332 }
9333 \@@_msg_new:nn { empty-preamble }
9334 {
9335     Empty-preamble.\\
9336     The-preamble-of-your~\@@_full_name_env:\ is-empty.\\
9337     This-error-is-fatal.
9338 }
9339 \@@_msg_new:nn { in-first-col }
9340 {
9341     Erroneous-use.\\
9342     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of-the-array.\\
9343     That-command-will-be-ignored.
9344 }

```

```

9345 \@@_msg_new:nn { in~last~col }
9346 {
9347   Erroneous~use.\\
9348   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9349   That~command~will~be~ignored.
9350 }
9351 \@@_msg_new:nn { in~first~row }
9352 {
9353   Erroneous~use.\\
9354   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9355   That~command~will~be~ignored.
9356 }
9357 \@@_msg_new:nn { in~last~row }
9358 {
9359   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9360   That~command~will~be~ignored.
9361 }
9362 \@@_msg_new:nn { caption~outside~float }
9363 {
9364   Key~caption~forbidden.\\
9365   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9366   environment.~This~key~will~be~ignored.
9367 }
9368 \@@_msg_new:nn { short~caption~without~caption }
9369 {
9370   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9371   However,~your~'short~caption'~will~be~used~as~'caption'.
9372 }
9373 \@@_msg_new:nn { double~closing~delimiter }
9374 {
9375   Double~delimiter.\\
9376   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9377   delimiter.~This~delimiter~will~be~ignored.
9378 }
9379 \@@_msg_new:nn { delimiter~after~opening }
9380 {
9381   Double~delimiter.\\
9382   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9383   delimiter.~That~delimiter~will~be~ignored.
9384 }
9385 \@@_msg_new:nn { bad~option~for~line~style }
9386 {
9387   Bad~line~style.\\
9388   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9389   is~'standard'.~That~key~will~be~ignored.
9390 }
9391 \@@_msg_new:nn { Identical~notes~in~caption }
9392 {
9393   Identical~tabular~notes.\\
9394   You~can't~put~several~notes~with~the~same~content~in~
9395   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9396   If~you~go~on,~the~output~will~probably~be~erroneous.
9397 }
9398 \@@_msg_new:nn { tabularnote~below~the~tabular }
9399 {
9400   \token_to_str:N \tabularnote\ forbidden\\
9401   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9402   of~your~tabular~because~the~caption~will~be~composed~below~
9403   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9404   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\

```

```

9405     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9406     no~similar~error~will~raised~in~this~document.
9407 }
9408 \@@_msg_new:nn { Unknown~key~for~rules }
9409 {
9410     Unknown~key.\\
9411     There~is~only~two~keys~available~here:~width~and~color.\\
9412     Your~key~'\l_keys_key_str'~will~be~ignored.
9413 }
9414 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9415 {
9416     Unknown~key.\\
9417     There~is~only~two~keys~available~here:~
9418     'empty'~and~'not-empty'.\\
9419     Your~key~'\l_keys_key_str'~will~be~ignored.
9420 }
9421 \@@_msg_new:nn { Unknown~key~for~rotate }
9422 {
9423     Unknown~key.\\
9424     The~only~key~available~here~is~'c'.\\
9425     Your~key~'\l_keys_key_str'~will~be~ignored.
9426 }
9427 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9428 {
9429     Unknown~key.\\
9430     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9431     It~you~go~on,~you~will~probably~have~other~errors. \\
9432     \c_@@_available_keys_str
9433 }
9434 {
9435     The~available~keys~are~(in~alphabetic~order):~
9436     ccommand,~
9437     color,~
9438     command,~
9439     dotted,~
9440     letter,~
9441     multiplicity,~
9442     sep~color,~
9443     tikz,~and~total~width.
9444 }
9445 \@@_msg_new:nnn { Unknown~key~for~xdots }
9446 {
9447     Unknown~key.\\
9448     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9449     \c_@@_available_keys_str
9450 }
9451 {
9452     The~available~keys~are~(in~alphabetic~order):~
9453     'color',~
9454     'horizontal~labels',~
9455     'inter',~
9456     'line~style',~
9457     'radius',~
9458     'shorten',~
9459     'shorten~end'~and~'shorten~start'.
9460 }
9461 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9462 {
9463     Unknown~key.\\
9464     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect~blocks'~
9465     (and~you~try~to~use~'\l_keys_key_str')\\

```

```

9466     That~key~will~be~ignored.
9467 }
9468 \@@_msg_new:nn { label~without~caption }
9469 {
9470     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9471     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9472 }
9473 \@@_msg_new:nn { W~warning }
9474 {
9475     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9476     (row~\int_use:N \c@iRow).
9477 }
9478 \@@_msg_new:nn { Construct~too~large }
9479 {
9480     Construct~too~large.\\
9481     Your~command~\token_to_str:N #1
9482     can't~be~drawn~because~your~matrix~is~too~small.\\
9483     That~command~will~be~ignored.
9484 }
9485 \@@_msg_new:nn { underscore~after~nicematrix }
9486 {
9487     Problem~with~'underscore'.\\
9488     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9489     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9490     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9491 }
9492 \@@_msg_new:nn { ampersand~in~light-syntax }
9493 {
9494     Ampersand~forbidden.\\
9495     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9496     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9497 }
9498 \@@_msg_new:nn { double~backslash~in~light-syntax }
9499 {
9500     Double~backslash~forbidden.\\
9501     You~can't~use~\token_to_str:N
9502     \\~to~separate~rows~because~the~key~'light-syntax'~
9503     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9504     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9505 }
9506 \@@_msg_new:nn { hlines~with~color }
9507 {
9508     Incompatible~keys.\\
9509     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9510     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9511     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9512     Your~key~will~be~discarded.
9513 }
9514 \@@_msg_new:nn { bad~value~for~baseline }
9515 {
9516     Bad~value~for~baseline.\\
9517     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9518     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9519     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9520     the~form~'line-i'.\\
9521     A~value~of~1~will~be~used.
9522 }
9523 \@@_msg_new:nn { detection~of~empty~cells }
9524 {
9525     Problem~with~'not~empty'\\

```

```

9526     For~technical~reasons,~you~must~activate~
9527     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9528     in~order~to~use~the~key~'\l_keys_key_str'.\\
9529     That~key~will~be~ignored.
9530 }
9531 \@@_msg_new:nn { siunitx-not-loaded }
9532 {
9533     siunitx-not-loaded\\
9534     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9535     That~error~is~fatal.
9536 }
9537 \@@_msg_new:nn { Invalid-name }
9538 {
9539     Invalid-name.\\
9540     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9541     \SubMatrix\ of~your~\@@_full_name_env:.\\
9542     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z] [A-Za-z0-9]*.\\
9543     This~key~will~be~ignored.
9544 }
9545 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9546 {
9547     Wrong~line.\\
9548     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9549     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9550     number~is~not~valid.~It~will~be~ignored.
9551 }
9552 \@@_msg_new:nn { Impossible-delimiter }
9553 {
9554     Impossible~delimiter.\\
9555     It's~impossible~to~draw~the~#1~delimiter~of~your~
9556     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9557     in~that~column.
9558     \bool_if:NT \l_@@_submatrix_slim_bool
9559     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9560     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9561 }
9562 \@@_msg_new:nnn { width-without-X-columns }
9563 {
9564     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9565     That~key~will~be~ignored.
9566 }
9567 {
9568     This~message~is~the~message~'width~without~X~columns'~
9569     of~the~module~'nicematrix'.~
9570     The~experimented~users~can~disable~that~message~with~
9571     \token_to_str:N \msg_redirect_name:nnn.\\
9572 }
9573
9574 \@@_msg_new:nn { key-multiplicity-with-dotted }
9575 {
9576     Incompatible~keys. \\
9577     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9578     in~a~'custom-line'.~They~are~incompatible. \\
9579     The~key~'multiplicity'~will~be~discarded.
9580 }
9581 \@@_msg_new:nn { empty-environment }
9582 {
9583     Empty~environment.\\
9584     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9585 }

```



```

9586 \@@_msg_new:nn { No~letter~and~no~command }
9587 {
9588   Erroneous~use.\\
9589   Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
9590   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9591   '~'ccommand'~(to~draw~horizontal~rules).\\
9592   However,~you~can~go~on.
9593 }
9594 \@@_msg_new:nn { Forbidden~letter }
9595 {
9596   Forbidden~letter.\\
9597   You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9598   It~will~be~ignored.
9599 }
9600 \@@_msg_new:nn { Several~letters }
9601 {
9602   Wrong~name.\\
9603   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9604   have~used~'\l_@@_letter_str').\\
9605   It~will~be~ignored.
9606 }
9607 \@@_msg_new:nn { Delimiter~with~small }
9608 {
9609   Delimiter~forbidden.\\
9610   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
9611   because~the~key~'small'~is~in~force.\\
9612   This~error~is~fatal.
9613 }
9614 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9615 {
9616   Unknown~cell.\\
9617   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9618   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
9619   can't~be~executed~because~a~cell~doesn't~exist.\\
9620   This~command~\token_to_str:N \line\ will~be~ignored.
9621 }
9622 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9623 {
9624   Duplicate~name.\\
9625   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
9626   in~this~\@@_full_name_env:.\\
9627   This~key~will~be~ignored.\\
9628   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9629     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9630 }
9631 {
9632   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9633   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9634 }
9635 \@@_msg_new:nn { r~or~l~with~preamble }
9636 {
9637   Erroneous~use.\\
9638   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9639   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9640   your~\@@_full_name_env:.\\
9641   This~key~will~be~ignored.
9642 }
9643 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9644 {
9645   Erroneous~use.\\
9646   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~

```

```

9647     the~array.~This~error~is~fatal.
9648 }
9649 \@@_msg_new:nn { bad~corner }
9650 {
9651     Bad~corner.\\
9652     #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9653     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9654     This~specification~of~corner~will~be~ignored.
9655 }
9656 \@@_msg_new:nn { bad~border }
9657 {
9658     Bad~border.\\
9659     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9660     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9661     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9662     also~use~the~key~'tikz'
9663     \IfPackageLoadedF { tikz }
9664         {~if~you~load~the~LaTeX~package~'tikz'}).\\
9665     This~specification~of~border~will~be~ignored.
9666 }
9667 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9668 {
9669     TikZ~not~loaded.\\
9670     You~can't~use~\token_to_str:N \TikzEveryCell\
9671     because~you~have~not~loaded~tikz.~
9672     This~command~will~be~ignored.
9673 }
9674 \@@_msg_new:nn { tikz~key~without~tikz }
9675 {
9676     TikZ~not~loaded.\\
9677     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9678     \Block'~because~you~have~not~loaded~tikz.~
9679     This~key~will~be~ignored.
9680 }
9681 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9682 {
9683     Erroneous~use.\\
9684     In~the~\@@_full_name_env:,~you~must~use~the~key~
9685     'last~col'~without~value.\\
9686     However,~you~can~go~on~for~this~time~
9687     (the~value~'\l_keys_value_tl'~will~be~ignored).
9688 }
9689 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9690 {
9691     Erroneous~use.\\
9692     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9693     'last~col'~without~value.\\
9694     However,~you~can~go~on~for~this~time~
9695     (the~value~'\l_keys_value_tl'~will~be~ignored).
9696 }
9697 \@@_msg_new:nn { Block~too~large~1 }
9698 {
9699     Block~too~large.\\
9700     You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
9701     too~small~for~that~block. \\
9702     This~block~and~maybe~others~will~be~ignored.
9703 }
9704 \@@_msg_new:nn { Block~too~large~2 }
9705 {
9706     Block~too~large.\\

```

```

9707 The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9708 \g_@@_static_num_of_col_int\
9709 columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9710 specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
9711 (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9712 This~block~and~maybe~others~will~be~ignored.
9713 }
9714 \@@_msg_new:nn { unknown~column~type }
9715 {
9716   Bad~column~type.\\
9717   The~column~type~'#1'~in~your~\@@_full_name_env:\
9718   is~unknown. \\
9719   This~error~is~fatal.
9720 }
9721 \@@_msg_new:nn { unknown~column~type~S }
9722 {
9723   Bad~column~type.\\
9724   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9725   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9726   load~that~package. \\
9727   This~error~is~fatal.
9728 }
9729 \@@_msg_new:nn { tabularnote~forbidden }
9730 {
9731   Forbidden~command.\\
9732   You~can't~use~the~command~\token_to_str:N\tabularnote\
9733   ~here.~This~command~is~available~only~in~
9734   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9735   the~argument~of~a~command~\token_to_str:N \caption\ included~
9736   in~an~environment~{table}. \\
9737   This~command~will~be~ignored.
9738 }
9739 \@@_msg_new:nn { borders~forbidden }
9740 {
9741   Forbidden~key.\\
9742   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9743   because~the~option~'rounded~corners'~
9744   is~in~force~with~a~non~zero~value.\\
9745   This~key~will~be~ignored.
9746 }
9747 \@@_msg_new:nn { bottomrule~without~booktabs }
9748 {
9749   booktabs~not~loaded.\\
9750   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9751   loaded~'booktabs'.\\
9752   This~key~will~be~ignored.
9753 }
9754 \@@_msg_new:nn { enumitem~not~loaded }
9755 {
9756   enumitem~not~loaded.\\
9757   You~can't~use~the~command~\token_to_str:N\tabularnote\
9758   ~because~you~haven't~loaded~'enumitem'.\\
9759   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9760   ignored~in~the~document.
9761 }
9762 \@@_msg_new:nn { tikz~without~tikz }
9763 {
9764   Tikz~not~loaded.\\
9765   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9766   loaded.~If~you~go~on,~that~key~will~be~ignored.
9767 }

```

```

9768 \@@_msg_new:nn { tikz-in-custom-line-without-tikz }
9769 {
9770   Tikz-not-loaded.\\
9771   You-have-used-the-key~'tikz'~in-the-definition-of~a~
9772   customized~line~(with~'custom-line')~but~tikz-is-not-loaded.~
9773   You-can-go-on-but-you-will-have-another-error-if-you-actually~
9774   use~that~custom-line.
9775 }

9776 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9777 {
9778   Tikz-not-loaded.\\
9779   You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9780   command~'\token_to_str:N\Block')~but~tikz-is-not-loaded.~
9781   That~key~will~be~ignored.
9782 }

9783 \@@_msg_new:nn { without-color-inside }
9784 {
9785   If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9786   \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9787   outside~\token_to_str:N \CodeBefore,~you~
9788   should~have~used~the~key~'color-inside'~in~your~\@@_full_name_env:.\\
9789   You-can-go-on-but-you-may~need~more~compilations.
9790 }

9791 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9792 {
9793   Erroneous-use.\\
9794   In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9795   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9796   The~key~'color'~will~be~discarded.
9797 }

9798 \@@_msg_new:nn { Wrong-last-row }
9799 {
9800   Wrong~number.\\
9801   You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9802   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9803   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9804   last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9805   without~value~(more~compilations~might~be~necessary).
9806 }

9807 \@@_msg_new:nn { Yet-in-env }
9808 {
9809   Nested~environments.\\
9810   Environments~of~nicematrix~can't~be~nested.\\
9811   This~error~is~fatal.
9812 }

9813 \@@_msg_new:nn { Outside-math-mode }
9814 {
9815   Outside~math~mode.\\
9816   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9817   (and~not~in~\token_to_str:N \vcenter).\\
9818   This~error~is~fatal.
9819 }

9820 \@@_msg_new:nn { One-letter~allowed }
9821 {
9822   Bad~name.\\
9823   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9824   It~will~be~ignored.
9825 }

9826 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9827 {

```

```

9828 Environment~{TabularNote}~forbidden.\\
9829 You-must-use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9830 but~*before*~the~\token_to_str:N \CodeAfter.\\
9831 This~environment~{TabularNote}~will~be~ignored.
9832 }
9833 \@@_msg_new:nn { varwidth~not~loaded }
9834 {
9835   varwidth~not~loaded.\\
9836   You-can't-use-the-column~type~'V'~because~'varwidth'~is~not~
9837   loaded.\\
9838   Your~column~will~behave~like~'p'.
9839 }
9840 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9841 {
9842   Unknown~key.\\
9843   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9844   \c_@@_available_keys_str
9845 }
9846 {
9847   The~available~keys~are~(in~alphabetic~order):~
9848   color,~
9849   dotted,~
9850   multiplicity,~
9851   sep-color,~
9852   tikz,~and~total-width.
9853 }
9854
9855 \@@_msg_new:nnn { Unknown~key~for~Block }
9856 {
9857   Unknown~key.\\
9858   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9859   \Block.\\ It~will~be~ignored. \\
9860   \c_@@_available_keys_str
9861 }
9862 {
9863   The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
9864   b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
9865   opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
9866   and~vlines.
9867 }
9868 \@@_msg_new:nnn { Unknown~key~for~Brace }
9869 {
9870   Unknown~key.\\
9871   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9872   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9873   It~will~be~ignored. \\
9874   \c_@@_available_keys_str
9875 }
9876 {
9877   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9878   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9879   right~shorten)~and~yshift.
9880 }
9881 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9882 {
9883   Unknown~key.\\
9884   The~key~'\l_keys_key_str'~is~unknown.\\
9885   It~will~be~ignored. \\
9886   \c_@@_available_keys_str
9887 }
9888 {
9889   The~available~keys~are~(in~alphabetic~order):~

```

```

9890     delimiters/color,~
9891     rules~(with~the~subkeys~'color'~and~'width'),~
9892     sub-matrix~(several~subkeys)~
9893     and~xdots~(several~subkeys).~
9894     The-latter-is-for-the-command~\token_to_str:N \line.
9895 }
9896 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9897 {
9898     Unknown~key.\\
9899     The-key~'\l_keys_key_str'~is~unknown.\\
9900     It~will~be~ignored. \\
9901     \c_@@_available_keys_str
9902 }
9903 {
9904     The-available-keys-are~(in~alphabetic~order):~
9905     create-cell-nodes,~
9906     delimiters/color~and~
9907     sub-matrix~(several~subkeys).
9908 }
9909 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9910 {
9911     Unknown~key.\\
9912     The-key~'\l_keys_key_str'~is~unknown.\\
9913     That~key~will~be~ignored. \\
9914     \c_@@_available_keys_str
9915 }
9916 {
9917     The-available-keys-are~(in~alphabetic~order):~
9918     'delimiters/color',~
9919     'extra-height',~
9920     'hlines',~
9921     'hvlines',~
9922     'left-xshift',~
9923     'name',~
9924     'right-xshift',~
9925     'rules'~(with~the~subkeys~'color'~and~'width'),~
9926     'slim',~
9927     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9928     and~'right-xshift').\\
9929 }
9930 \@@_msg_new:nnn { Unknown~key~for~notes }
9931 {
9932     Unknown~key.\\
9933     The-key~'\l_keys_key_str'~is~unknown.\\
9934     That~key~will~be~ignored. \\
9935     \c_@@_available_keys_str
9936 }
9937 {
9938     The-available-keys-are~(in~alphabetic~order):~
9939     bottomrule,~
9940     code-after,~
9941     code-before,~
9942     detect-duplicates,~
9943     enumitem-keys,~
9944     enumitem-keys-para,~
9945     para,~
9946     label-in-list,~
9947     label-in-tabular~and~
9948     style.
9949 }
9950 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9951 {

```

```

9952     Unknown~key.\\
9953     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9954     \token_to_str:N \RowStyle. \\
9955     That~key~will~be~ignored. \\
9956     \c_@@_available_keys_str
9957   }
9958   {
9959     The~available~keys~are~(in~alphabetic~order):~
9960     bold,~
9961     cell-space-top-limit,~
9962     cell-space-bottom-limit,~
9963     cell-space-limits,~
9964     color,~
9965     fill~(alias:~rowcolor),~
9966     nb-rows~and~
9967     opacity.
9968   }
9969   \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9970   {
9971     Unknown~key.\\
9972     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9973     \token_to_str:N \NiceMatrixOptions. \\
9974     That~key~will~be~ignored. \\
9975     \c_@@_available_keys_str
9976   }
9977   {
9978     The~available~keys~are~(in~alphabetic~order):~
9979     &-in-blocks,~
9980     allow-duplicate-names,~
9981     ampersand-in-blocks,~
9982     caption-above,~
9983     cell-space-bottom-limit,~
9984     cell-space-limits,~
9985     cell-space-top-limit,~
9986     code-for-first-col,~
9987     code-for-first-row,~
9988     code-for-last-col,~
9989     code-for-last-row,~
9990     corners,~
9991     custom-key,~
9992     create-extra-nodes,~
9993     create-medium-nodes,~
9994     create-large-nodes,~
9995     custom-line,~
9996     delimiters~(several~subkeys),~
9997     end-of-row,~
9998     first-col,~
9999     first-row,~
10000    hlines,~
10001    hvlines,~
10002    hvlines-except-borders,~
10003    last-col,~
10004    last-row,~
10005    left-margin,~
10006    light-syntax,~
10007    light-syntax-expanded,~
10008    matrix/columns-type,~
10009    no-cell-nodes,~
10010    notes~(several~subkeys),~
10011    nullify-dots,~
10012    pgf-node-code,~
10013    renew-dots,~
10014    renew-matrix,~

```

```

10015     respect-arraystretch,~
10016     rounded-corners,~
10017     right-margin,~
10018     rules~(with~the~subkeys~'color'~and~'width'),~
10019     small,~
10020     sub-matrix~(several~subkeys),~
10021     vlines,~
10022     xdots~(several~subkeys).
10023 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

10024 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10025 {
10026   Unknown~key.\\
10027   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10028   \{NiceArray\}. \\
10029   That~key~will~be~ignored. \\
10030   \c_@@_available_keys_str
10031 }
10032 {
10033   The~available~keys~are~(in~alphabetic~order):~
10034   &~in~blocks,~
10035   ampersand~in~blocks,~
10036   b,~
10037   baseline,~
10038   c,~
10039   cell-space-bottom-limit,~
10040   cell-space-limits,~
10041   cell-space-top-limit,~
10042   code~after,~
10043   code~for~first~col,~
10044   code~for~first~row,~
10045   code~for~last~col,~
10046   code~for~last~row,~
10047   color~inside,~
10048   columns~width,~
10049   corners,~
10050   create~extra~nodes,~
10051   create~medium~nodes,~
10052   create~large~nodes,~
10053   extra~left~margin,~
10054   extra~right~margin,~
10055   first~col,~
10056   first~row,~
10057   hlines,~
10058   hvlines,~
10059   hvlines~except~borders,~
10060   last~col,~
10061   last~row,~
10062   left~margin,~
10063   light~syntax,~
10064   light~syntax~expanded,~
10065   name,~
10066   no~cell~nodes,~
10067   nullify~dots,~
10068   pgf~node~code,~
10069   renew~dots,~
10070   respect-arraystretch,~
10071   right-margin,~
10072   rounded-corners,~
10073   rules~(with~the~subkeys~'color'~and~'width'),~
10074   small,~
10075   t,~

```



```

10076     vlines,~
10077     xdots/color,~
10078     xdots/shorten-start,~
10079     xdots/shorten-end,~
10080     xdots/shorten-and~
10081     xdots/line-style.
10082 }

```

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`).

```

10083 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10084 {
10085   Unknown~key.\\
10086   The~key~'\l_keys_key_str'~is~unknown~for~the~
10087   \@@_full_name_env:. \\
10088   That~key~will~be~ignored. \\
10089   \c_@@_available_keys_str
10090 }
10091 {
10092   The~available~keys~are~(in~alphabetic~order):~
10093   &~in~blocks,~
10094   ampersand~in~blocks,~
10095   b,~
10096   baseline,~
10097   c,~
10098   cell-space-bottom-limit,~
10099   cell-space-limits,~
10100   cell-space-top-limit,~
10101   code-after,~
10102   code-for-first-col,~
10103   code-for-first-row,~
10104   code-for-last-col,~
10105   code-for-last-row,~
10106   color-inside,~
10107   columns-type,~
10108   columns-width,~
10109   corners,~
10110   create-extra-nodes,~
10111   create-medium-nodes,~
10112   create-large-nodes,~
10113   extra-left-margin,~
10114   extra-right-margin,~
10115   first-col,~
10116   first-row,~
10117   hlines,~
10118   hvlines,~
10119   hvlines-except-borders,~
10120   l,~
10121   last-col,~
10122   last-row,~
10123   left-margin,~
10124   light-syntax,~
10125   light-syntax-expanded,~
10126   name,~
10127   no-cell-nodes,~
10128   nullify-dots,~
10129   pgf-node-code,~
10130   r,~
10131   renew-dots,~
10132   respect-arraystretch,~
10133   right-margin,~
10134   rounded-corners,~
10135   rules~(with~the~subkeys~'color'~and~'width'),~
10136   small,~

```

```

10137     t,~
10138     vlines,~
10139     xdots/color,~
10140     xdots/shorten-start,~
10141     xdots/shorten-end,~
10142     xdots/shorten-and~
10143     xdots/line-style.
10144 }
10145 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10146 {
10147     Unknown~key.\\
10148     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10149     \{NiceTabular\}. \\
10150     That~key~will~be~ignored. \\
10151     \c_@@_available_keys_str
10152 }
10153 {
10154     The~available~keys~are~(in~alphabetic~order):~
10155     &~in~blocks,~
10156     ampersand~in~blocks,~
10157     b,~
10158     baseline,~
10159     c,~
10160     caption,~
10161     cell-space-bottom-limit,~
10162     cell-space-limits,~
10163     cell-space-top-limit,~
10164     code-after,~
10165     code-for-first-col,~
10166     code-for-first-row,~
10167     code-for-last-col,~
10168     code-for-last-row,~
10169     color-inside,~
10170     columns-width,~
10171     corners,~
10172     custom-line,~
10173     create-extra-nodes,~
10174     create-medium-nodes,~
10175     create-large-nodes,~
10176     extra-left-margin,~
10177     extra-right-margin,~
10178     first-col,~
10179     first-row,~
10180     hlines,~
10181     hvlines,~
10182     hvlines-except-borders,~
10183     label,~
10184     last-col,~
10185     last-row,~
10186     left-margin,~
10187     light-syntax,~
10188     light-syntax-expanded,~
10189     name,~
10190     no-cell-nodes,~
10191     notes~(several~subkeys),~
10192     nullify-dots,~
10193     pgf-node-code,~
10194     renew-dots,~
10195     respect-arraystretch,~
10196     right-margin,~
10197     rounded-corners,~
10198     rules~(with~the~subkeys~'color'~and~'width'),~
10199     short-caption,~

```

```

10200     t,~
10201     tabularnote,~
10202     vlines,~
10203     xdots/color,~
10204     xdots/shorten-start,~
10205     xdots/shorten-end,~
10206     xdots/shorten-and~
10207     xdots/line-style.
10208 }

10209 \@@_msg_new:nnn { Duplicate~name }
10210 {
10211     Duplicate~name.\\
10212     The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10213     the~same~environment~name~twice.~You~can~go~on,~but,~
10214     maybe,~you~will~have~incorrect~results~especially~
10215     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10216     message~again,~use~the~key~'allow-duplicate-names'~in~
10217     '\token_to_str:N \NiceMatrixOptions'.\\
10218     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10219     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10220 }
10221 {
10222     The~names~already~defined~in~this~document~are:~
10223     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10224 }

10225 \@@_msg_new:nn { Option~auto~for~columns-width }
10226 {
10227     Erroneous~use.\\
10228     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10229     That~key~will~be~ignored.
10230 }

10231 \@@_msg_new:nn { NiceTabularX~without~X }
10232 {
10233     NiceTabularX~without~X.\\
10234     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10235     However,~you~can~go~on.
10236 }

10237 \@@_msg_new:nn { Preamble~forgotten }
10238 {
10239     Preamble~forgotten.\\
10240     You~have~probably~forgotten~the~preamble~of~your~
10241     \@@_full_name_env:. \\
10242     This~error~is~fatal.
10243 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Collecting options	3
3	Technical definitions	4
4	Parameters	8
5	The command <code>\tabularnote</code>	18
6	Command for creation of rectangle nodes	23
7	The options	24
8	Important code used by <code>{NiceArrayWithDelims}</code>	35
9	The <code>\CodeBefore</code>	49
10	The environment <code>{NiceArrayWithDelims}</code>	53
11	Construction of the preamble of the array	58
12	The redefinition of <code>\multicolumn</code>	73
13	The environment <code>{NiceMatrix}</code> and its variants	91
14	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
15	After the construction of the array	93
16	We draw the dotted lines	100
17	The actual instructions for drawing the dotted lines with <code>Tikz</code>	113
18	User commands available in the new environments	119
19	The command <code>\line</code> accessible in code-after	125
20	The command <code>\RowStyle</code>	127
21	Colors of cells, rows and columns	129
22	The vertical and horizontal rules	141
23	The empty corners	156
24	The environment <code>{NiceMatrixBlock}</code>	158
25	The extra nodes	160
26	The blocks	164
27	How to draw the dotted lines transparently	188
28	Automatic arrays	188
29	The redefinition of the command <code>\dotfill</code>	190
30	The command <code>\diagbox</code>	190

31	The keyword <code>\CodeAfter</code>	192
32	The delimiters in the preamble	192
33	The command <code>\SubMatrix</code>	194
34	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	202
35	The command <code>TikzEveryCell</code>	205
36	The command <code>\ShowCellNames</code>	207
37	We process the options at package loading	208
38	About the package underscore	210
39	Error messages of the package	210